Giới thiệu
MỤC ĐÍCH Chương này trình bày một cái nhìn bao quát về cơ sở dữ liệu (CSDL/DB), về hệ quản trị cơ sở dữ liệu (HQTCSDL/DBMS) và về hệ cơ sở dữ liệu (HCSDL/DBS). Các đòi hỏi khi xây dựng một HQTCSDL đó cũng chính là những chức năng mà một HCSDL cần phải có. Một khái niệm quan trọng là khái niệm giao dịch (Transaction). Các tính chất một giao dịch phải có để đảm bảo một HQTCSDL, được xây dựng trên HCSDL tương ứng, trong suốt quá trình hoạt động sẽ luôn cho một CSDL tin cậy (dữ liệu luôn nhất quán). Quản trị giao dịch nhằm đảm bảo mọi giao dịch trong hệ thống có các tính chất mà một giao dịch phải có. Một điều cần chú ý là trong các tính chất của một giao dịch, tính chất nhất quán trước hết phải được đảm bảo bởi người lập trình-người viết ra giao dịch. YÊU CẦU Hiểu các khái niệm. Hiểu các vấn đề đặt ra khi xây dựng một HQTCSDL: thiết kế CSDL, đảm bảo tính nhất quán của CSDL trong suốt cuộc sống của nó, nền tảng phần cứng trên đó một HQTCSDL được xây dựng. Hiểu cấu trúc hệ thống tổng thể Hiểu vai trò của các người sử dụng hệ thống. MỘT SỐ KHÁI NIỆM • Một cơ sở dữ liệu (CSDL/ DB: DataBase) là một tập hợp các tập tin có liên quan với nhau, được thiết kế nhằm làm giảm thiểu sự lặp lại dữ liệu. • Một hệ quản trị cơ sở dữ liệu (HQTCSDL/ DBMS: DataBase Management System) là một hệ thống gồm một CSDL và các thao tác trên CSDL đó, được thiết kế trên một nền tảng phần cứng, phần mềm và với một kiến trúc nhất định. • Một hệ cơ sở dữ liệu (HCSDL/ DBS: DataBase System) là một phần mềm cho phép xây dựng một HQTCSDL.

HỆ CƠ SỞ DỮ LIỆU

Một số điểm bất lợi chính của việc lưu giữ thông tin có tổ chức trong hệ thống xử lý file thông thường:

- Dư thừa dữ liệu và tính không nhất quán (Data redundancy and inconsistency): Do các file và các trình ứng dụng được tạo ra bởi các người lập trình khác nhau, nên các file có định dạng khác nhau, các chương trình được viết trong các ngôn ngữ lập trình khác nhau, cùng một thông tin có thể được lưu giữ trong các file khác nhau. Tính

không thống nhất và dư thừa này sẽ làm tăng chi phí truy xuất và lưu trữ, hơn nũa, nó sẽ dẫn đến tính không nhất quán của dữ liệu: các bản sao của cùng một dữ liệu có thể không nhất quán.

- Khó khăn trong việc truy xuất dữ liệu: Môi trường của hệ thống xử lý file thông thường không cung cấp các công cụ cho phép truy xuất thông tin một cách hiệu quả và thuận lợi.

- Sự cô lập dữ liệu (Data isolation): Các giá trị dữ liệu được lưu trữ trong cơ sở dữ liệu phải thoả mãn một số các ràng buộc về tính nhất quán của dữ liệu (ràng buộc nhất quán/consistency contraints). Trong hệ thống xử lý file thông thường, rất khó khăn trong việc thay đổi các chương trình để thoả mãn các yêu cầu thay đổi ràng buộc. Vấn đề trở nên khó khăn hơn khi các ràng buộc liên quan đến các hạng mục dữ liệu nằm trong các file khác nhau.

- Các vấn đề về tính nguyên tử (Atomicity problems): Tính nguyên tử của một hoạt động (giao dịch) là: hoặc nó được hoàn tất trọn vẹn hoặc không có gì cả. Điều này có nghĩa là một hoạt động (giao dịch) chỉ làm thay đổi các dữ liệu bền vững khi nó đã hoàn tất (kết thúc thành công) nếu không, giao dịch không để lại một dấu vết nào trên CSDL. Trong hệ thống xử lý file thông thường khó đảm bảo được tính chất này.

- Tính bất thường trong truy xuất cạnh tranh: Một hệ thống cho phép nhiều người sử dụng cập nhật dữ liệu đồng thời, có thể dẫn đến kết quả là dữ liệu không nhất quán. Điều này đòi hỏi một sự giám sát. Hệ thống xử lý file thông thường không cung cấp chức năng này.

- Vấn đề an toàn (Security problems): một người sử dụng hệ cơ sở dữ liệu không cần thiết và cũng không có quyền truy xuất tất cả các dữ liệu. Vấn đề này đòi hỏi hệ thống phải đảm bảo được tính phân quyền, chống truy xuất trái phép ...

Các bất lợi nêu trên đã gợi mở sự phát triển các DBMS. Phần sau của giáo trình sẽ đề cập đến các quan niệm và các thuật toán được sử dụng

để phát triển một hệ cơ sở dữ liệu nhằm giải quyết các vấn đề nêu trên. Một số khái niệm

GÓC NHÌN DỮ LIỆU

Tính hiệu quả của hệ thống đòi hỏi phải thiết kế các cấu trúc dữ liệu phức tạp để biểu diễn dữ liệu trong cơ sở dữ liệu. Các nhà phát triển che dấu sự phức tạp này thông qua các mức trừu tượng nhằm đơn giản hóa sự trao đổi của người sử dụng với hệ thống:

- Mức vật lý (Physical level): Mức thấp nhất của sự trừu tượng, mô tả dữ liệu hiện được lưu trữ thế nào. Ở mức này, cấu trúc dữ liệu mức thấp, phức tạp được mô tả chi tiết.

- Mức luận lý (Logical level): Mức kế cao hơn về sự trừu tượng, mô tả dữ liệu gì được lưu trữ trong cơ sở dữ liệu và các mối quan hệ gì giữa các dữ liệu này. Mức logic của sự trừu tượng được dùng bởi các người quản trị cơ sở dữ liệu.

- Mức view (view level): Mức cao nhất của sự trừu tượng, mô tả chỉ một phần của cơ sở dữ liệu toàn thể. Một người sử dụng cơ sở dữ liệu liên quan đến chỉ một bộ phận của cơ sở dữ liệu. Như vậy sự trao đổi của họ với hệ thống được làm đơn giản bởi việc định nghĩa view. Hệ thống có thể cung cấp nhiều mức view đối với cùng một cơ sở dữ liệu.

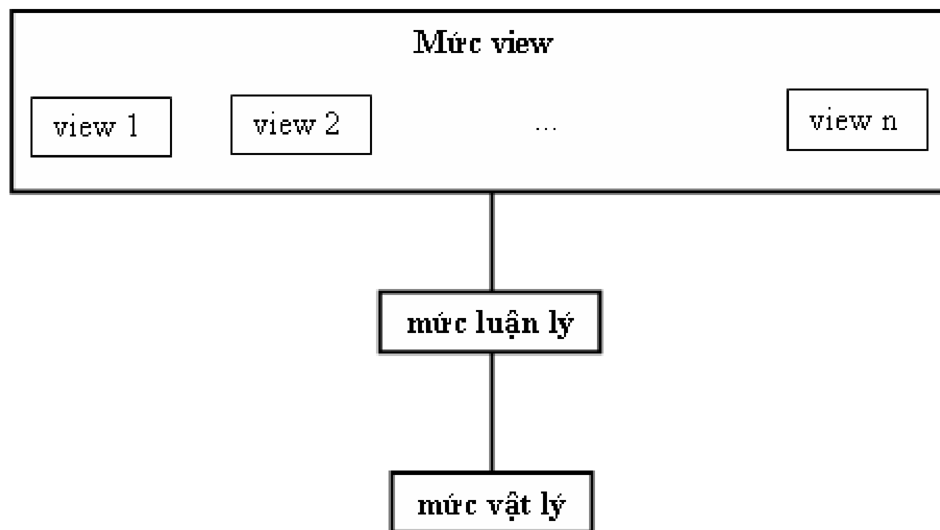


Figure 1

- Thể hiện và sơ đồ (instances and schemas): Tập hợp các thông tin được lưu trữ trong cơ sở dữ liệu tại một thời điểm được gọi là một thể hiện (instance) của cơ sở dữ liệu. Thiết kế tổng thể của cơ sở dữ liệu được gọi là sơ đồ (schema).

Một hệ cơ sở dữ liệu có một vài sơ đồ, được phân tương ứng với các mức trừu tượng. ở mức thấp nhất là sơ đồ vật lý (physical schema), ở mức trung gian là sơ đồ luận lý (logical schema), ở mức cao nhất là sơ đồ con (subschema). Nói chung một hệ cơ sở dữ liệu hỗ trợ một sơ đồ vật lý, một sơ đồ luận lý và một vài sơ đồ con.

- Khả năng sửa đổi một định nghĩa ở một mức không ảnh hưởng một định nghĩa sơ đồ ở mức cao hơn được gọi là sự độc lập dữ liệu (data independence). Có hai mức độc lập dữ liệu:

- Độc lập dữ liệu vật lý (Physical data independence) là khả năng sửa đổi sơ đồ vật lý không làm cho các chương trình ứng dụng phải viết lại. Các sửa đổi ở mức vật lý là cần thiết để cải thiện hiệu năng.
- Độc lập dữ liệu luận lý (Logical data independence) là khả năng sửa đổi sơ đồ luận lý không làm cho các chương trình ứng dụng phải viết lại. Các sửa đổi ở mức luận lý là cần thiết khi cấu trúc luận lý của cơ sở dữ liệu bị thay thế.

MÔ HÌNH DỮ LIỆU

Nằm dưới cấu trúc của một cơ sở dữ liệu là mô hình dữ liệu: một bộ các công cụ quan niệm để mô tả dữ liệu, quan hệ dữ liệu, ngữ nghĩa dữ liệu và các ràng buộc nhất quán. Có ba nhóm mô hình: Các mô hình luận lý dựa trên đối tượng (Object-based logical models), các mô hình luận lý dựa trên mẩu tin (record-based logical models), các mô hình vật lý (physical models).

- Các mô hình luận lý dựa trên đối tượng được dùng mô tả dữ liệu ở mức luận lý và mức view. Chúng được đặc trưng bởi việc chúng cung cấp khả năng cấu trúc linh hoạt và cho phép các ràng buộc dữ liệu được xác định một cách tường minh. Dưới đây là một vài mô hình được biết rộng rãi: Mô hình thực thể - quan hệ (entity-relationship model), mô hình hướng đối tượng (object-oriented model), mô hình dữ liệu ngữ nghĩa (semantic data model), mô hình dữ liệu hàm (function data model).
- Các mô hình luận lý dựa trên mẩu tin được dùng để miêu tả dữ liệu ở mức luận lý hay mức view. Chúng được dùng để xác định cấu trúc luận lý toàn thể của cơ sở dữ liệu và cung cấp sự mô tả mức cao hơn việc thực hiện. Cơ sở dữ liệu được cấu trúc ở dạng mẩu tin định dạng cố định (fixed format record): mỗi mẩu tin xác định một số cố định các trường, mối trường thường có độ dài cố định. Một vài mô hình được biết rộng rãi là: Mô hình quan hệ, mô hình mạng, mô hình phân cấp.
- Mô hình dữ liệu vật lý được dùng để mô tả dữ liệu ở mức thấp nhất. Hai mô hình dữ liệu vật lý được biết rộng rãi nhất là mô hình hợp nhất (unifying model) và mô hình khung-bộ nhớ (frame-memory model).

NGÔN NGỮ CƠ SỞ DỮ LIỆU

Một hệ cơ sở dữ liệu cung cấp hai kiểu ngôn ngữ khác nhau: một để xác định sơ đồ cơ sở dữ liệu, một để biểu diễn các vấn tin cơ sở dữ liệu và cập nhật.

- Ngôn ngữ định nghĩa dữ liệu (Data Definition Language: DDL) cho phép định nghĩa sơ đồ cơ sở dữ liệu. Kết quả biên dịch các lệnh của DDL là tập hợp các bảng được lưu trữ trong một file đặc biệt được gọi là tự điển dữ liệu (data dictionary) hay thư mục dữ liệu (data directory). Tự điển dữ liệu là một file chứa metadata. File này được tra cứu trước khi dữ liệu hiện hành được đọc hay sửa đổi. Cấu trúc lưu trữ và phương pháp truy cập được sử dụng bởi hệ cơ sở dữ liệu được xác định bởi một tập hợp các định nghĩa trong một kiểu đặc biệt của DDL được gọi là ngôn ngữ định nghĩa và lưu trữ dữ liệu (data storage and definition language). Kết quả biên dịch của các định nghĩa này là một tập hợp các chỉ thị xác định sự thực hiện chi tiết của các sơ đồ cơ sở dữ liệu (thường được che dấu).
- Ngôn ngữ thao tác dữ liệu (Data manipulation language: DML) là ngôn ngữ cho phép người sử dụng truy xuất hoặc thao tác dữ liệu. Có hai kiểu ngôn ngữ thao tác dữ liệu: DML thủ tục (procedural DML) yêu cầu người sử dụng đặc tả dữ liệu nào cần và làm thế nào để nhận được nó. DML không thủ tục (Nonprocedural DML) yêu cầu người sử dụng đặc tả dữ liệu nào cần nhưng không cần đặc tả làm thế nào để nhận được nó. Một vấn tin (query) là một lệnh yêu cầu tìm lại dữ liệu (information retrieval). Phần ngôn ngữ DML liên quan đến sự tìm lại thông tin được gọi là ngôn ngữ vấn tin (query language).

QUẢN TRỊ GIAO DỊCH

Thông thường, một số thao tác trên cơ sở dữ liệu tạo thành một đơn vị logic công việc. Ta hãy xét ví dụ chuyển khoản, trong đó một số tiền x được chuyển từ tài khoản A (A:=A-x) sang một tài khoản B (B:=B+x). Một yếu tố cần thiết là cả hai thao tác này hoặc cùng xảy ra hoặc không hoạt động nào xảy ra cả. Việc chuyển khoản phải xảy ra trong tính toàn thể của nó hoặc không. Đòi hỏi toàn thể-hoặc-không này được gọi là tính nguyên tử (atomicity). Một yếu tố cần thiết khác là sự thực hiện việc chuyển khoản bảo tồn tính nhất quán của cơ sở dữ liệu: giá trị của tổng A + B phải được bảo tồn. Đòi hỏi về tính chính xác này được gọi là tính nhất quán (consistency). Cuối cùng, sau khi thực hiện thành công hoạt

động chuyển khoản, các giá trị của các tài khoản A và B phải bền vững cho dù có thể có sự cố hệ thống. Đòi hỏi về tính bền vững này được gọi là tính lâu bền (durability).

Một giao dịch là một tập các hoạt động thực hiện chỉ một chức năng logic trong một ứng dụng cơ sở dữ liệu. Mỗi giao dịch là một đơn vị mang cả tính nguyên tử lẫn tính nhất quán. Như vậy, các giao dịch phải không được vi phạm bất kỳ ràng buộc nhất quán nào: Nếu cơ sở dữ liệu là nhất quán khi một giao dịch khởi động thì nó cũng phải là nhất quán khi giao dịch kết thúc thành công. Tuy nhiên, trong khi đang thực hiện giao dịch, phải cho phép sự không nhất quán tạm thời. Sự không nhất quán tạm thời này tuy là cần thiết nhưng lại có thể dẫn đến các khó khăn nếu xảy ra sự cố.

Trách nhiệm của người lập trình là xác định đúng đắn các giao dịch sao cho mỗi một bảo tồn tính nhất quán của cơ sở dữ liệu.

Đảm bảo tính nguyên tử và tính lâu bền là trách nhiệm của hệ cơ sở dữ liệu nói chung và của thành phần quản trị giao dịch (transaction-management component) nói riêng. Nếu không có sự cố, tất cả giao dịch hoàn tất thành công và tính nguyên tử được hoàn thành dễ dàng. Tuy nhiên, do sự hiện diện của các sự cố, một giao dich có thể không hoàn tất thành công sự thực hiện của nó. Nếu tính nguyên tử được đảm bảo, một giao dịch thất bại không gây hiệu quả đến trạng thái của cơ sở dữ liệu. Như vậy, cơ sở dữ liệu phải được hoàn lại trạng thái của nó trước khi giao dịch bắt đầu. Hệ cơ sở dữ liệu phải có trách nhiệm phát hiện sự cố hệ thống và trả lại cơ sở dữ liệu về trạng thái trước khi xảy ra sự cố.

Khi một số giao dịch cạnh tranh cập nhật cơ sở dữ liệu, tính nhất quán của dữ liệu có thể không được bảo tồn, ngay cả khi mỗi giao dịch là chính xác. Bộ quản trị điều khiển cạnh tranh (concurency-control manager) có trách nhiệm điều khiển các trao đổi giữa các giao dịch cạnh tranh để đảm bảo tính thống nhất của CSDL.

QUẢN TRỊ LƯU TRỮ

Các CSDL đòi hỏi một khối lượng lớn không gian lưu trữ, có thể lên đến nhiều terabytes (1 terabyte=103 Gigabytes=106 Megabytes). Các thông tin phải được lưu trữ trên lưu trữ ngoài (đĩa). Dữ liệu được di chuyển giữa lưu trữ đĩa và bộ nhớ chính khi cần thiết. Do việc di chuyển dữ liệu từ và lên đĩa tương đối chậm so với tốc độ của đơn vị xử lý trung tâm, điều này ép buộc hệ CSDL phải cấu trúc dữ liệu sao cho tối ưu hóa nhu cầu di chuyển dữ liệu giữa đĩa và bộ nhớ chính.

Mục đích của một hệ CSDL là làm đơn giản và dễ dàng việc truy xuất dữ liệu. Người sử dụng hệ thống có thể không cần quan tâm đến chi tiết vật lý của sự thực thi hệ thống. Phần lớn họ chỉ quan đến hiệu năng của hệ thống (thời gian trả lời một câu vấn tin ...).

Bộ quản trị lưu trữ (storage manager) là một module chương trình cung cấp giao diện giữa dữ liệu mức thấp được lưu trữ trong CSDL với các chương trình ứng dụng và các câu vấn tin được đệ trình cho hệ thống. Bộ quản trị lưu trữ có trách nhiệm trao đổi với bộ quản trị file (file manager). Dữ liệu thô được lưu trữ trên đĩa sử dụng hệ thống file (file system), hệ thống này thường được cung cấp bởi hệ điều hành. Bộ quản trị lưu trữ dịch các câu lệnh DML thành các lệnh của hệ thống file mức thấp. Như vậy, bộ quản trị lưu trữ có nhiệm vụ lưu trữ, tìm lại và cập nhật dữ liệu trong CSDL.

NHÀ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Một trong các lý do chính đối với việc sử dụng DBMS là có sự điều khiển trung tâm cho cả dữ liệu lẫn các chương trình truy cập các dữ liệu này. Người điều khiển trung tâm trên toàn hệ thống như vậy gọi là nhà quản trị cơ sở dữ liệu (DataBase Administrator - DBA). Các chức năng của DBA như sau:

- Định nghĩa sơ đồ: DBA tạo ra sơ đồ CSDL gốc bằng cách viết một tập các định nghĩa mà nó sẽ được dịch bởi trình biên dịch DDL thành một tập các bảng được lưu trữ thường trực trong tự điển dữ liệu.

- Định nghĩa cấu trúc lưu trữ và phương pháp truy xuất: DBA tạo ra một cấu trúc lưu trữ thích hợp và các phương pháp truy xuất bằng cách viết một tập hợp các định nghĩa mà nó sẽ được dịch bởi trình biên dịch lưu trữ dữ liệu và ngôn ngữ định nghĩa dữ liệu.
- Sửa đổi sơ đồ và tổ chức vật lý
- Cấp quyền truy xuất dữ liệu: Việc cấp các dạng quyền truy cập khác nhau cho phép DBA điều hoà những phần của CSDL mà nhiều người có thể truy xuất. Thông tin về quyền được lưu giữ trong một cấu trúc hệ thống đặc biệt, nó được tham khảo bởi hệ CSDL mỗi khi có sự truy xuất dữ liệu của hệ thống.
- Đặc tả ràng buộc toàn vẹn (integrity-contraint): Các giá trị dữ liệu được lưu trữ trong CSDL phải thoả mãn một số các ràng buộc nhất quán nhất định. Ví dụ số giờ làm việc của một nhân viên trong một tuần không thể vượt quá một giới hạn 80 giờ chẳng hạn. Một ràng buộc như vậy phải được đặc tả một cách tường minh bởi DBA. Các ràng buộc toàn vẹn được lưu giữ trong một cấu trúc hệ thống đặc biệt được tham khảo bởi hệ CSDL mỗi khi có sự cập nhật dữ liệu.

NGƯỜI SỬ DỤNG CSDL

Mục đích đầu tiên của hệ CSDL là cung cấp một môi trường để tìm lại thông tin và lưu thông tin trong CSDL. Các người sử dụng cơ sở dữ liệu được phân thành bốn nhóm tuỳ theo cách thức họ trao đổi với hệ thống.

- Các người lập trình ứng dụng: Là nhà chuyên môn máy tính người trao đổi với hệ thống thông qua các lời gọi DML được nhúng trong một chương trình được viết trong một ngôn ngữ chủ - host language (Pascal, C, Cobol ...). Các chương trình này thường được tham khảo như các chương trình ứng dụng. Vì cú pháp DML thường rất khác với cú pháp của ngôn ngữ chủ, các lời gọi DML thường được bắt đầu bởi một ký tự đặc biệt như vậy mã thích hợp mới có thể được sinh. Một bộ tiền xử lý đặc biệt, được gọi là tiền biên dịch (precompiler) DML, chuyển các lệnh DML thành các lời gọi thủ tục

chuẩn trong ngôn ngữ chủ. Bộ biên dịch ngôn ngữ chủ sẽ sinh mã đối tượng thích hợp. Có những ngôn ngữ lập trình phối hợp cấu trúc điều khiển của các ngôn ngữ giống như Pascal với cấu trúc điều khiển để thao tác đối tượng CSDL. Các ngôn ngữ này (đôi khi được gọi là ngôn ngữ thế hệ thứ tư) thường bao gồm các đặc điểm đặc biệt để làm dễ dàng việc sinh các dạng và hiển thị dữ liệu trên màn hình.

- Các người sử dụng thành thạo (Sophisticated users): Trao đổi với hệ thống không qua viết trình. Thay vào đó họ đặt ra các yêu cầu của họ trong ngôn ngữ truy vấn CSDL (Database query language). Mỗi câu vấn tin như vậy được đệ trình cho bộ xử lý vấn tin, chức năng của bộ xử lý vấn tin là "dịch" các lệnh DML thành các chỉ thị mà bộ quản trị lưu trữ hiểu. Các nhà phân tích đệ trình các câu vấn tin thăm dò dữ liệu trong cơ sở dữ liệu thuộc vào phạm trù này.

- Các người sử dụng chuyên biệt (Specialized users): Là các người sử dụng thành thạo, họ viết các ứng dụng CSDL chuyên biệt không nằm trong khung xử lý dữ liệu truyền thống. Trong đó, phải kể đến các hệ thống thiết kế được trợ giúp bởi máy tính (computer-aided design systems), Cơ sở tri thức (knowledge-base) và hệ chuyên gia (expert systems), các hệ thống lưu trữ dữ liệu với kiểu dữ liệu phức tạp (dữ liệu đồ họa, hình ảnh, âm thanh) và các hệ thống mô hình môi trường (environment-modeling systems)

- Các người sử dụng ngây thơ (Naive users): là các người sử dụng không thành thạo, họ trao đổi với hệ thống bởi cầu dẫn một trong các chương trình ứng dụng thường trực đã được viết sẵn.

CẤU TRÚC HỆ THỐNG TỔNG THỂ

Một hệ CSDL được phân thành các module, mỗi một thực hiện một trách nhiệm trong hệ thống tổng thể. Một số chức năng của hệ CSDL có thể được cung cấp bởi hệ điều hành. Trong hầu hết các trường hợp, hệ điều hành chỉ cung cấp các dịch vụ cơ sở nhất, hệ CSDL phải xây dựng trên

cơ sở đó. Như vậy, thiết kế hệ CSDL phải xem xét đến giao diện giữa hệ CSDL và hệ điều hành.

Các thành phần chức năng của hệ CSDL có thể được chia thành các thành phần xử lý vấn tin (query processor components) và các thành phần quản trị lưu trữ (storage manager components).

Các thành phần xử lý vấn tin gồm:

- Trình biên dịch DML (DML compiler): dịch các lệnh DML trong một ngôn ngữ vấn tin thành các chỉ thị mức thấp mà engine định giá vấn tin (query evaluation engine) có thể hiểu. Hơn nữa, Trình biên dịch DML phải biến đổi một yêu cầu của người sử dụng thành một đích tương đương nhưng ở dạng hiệu quả hơn có nghĩa là tìm một chiến lược tốt để thực hiện câu vấn tin.

- Trình tiền biên dịch DML nhúng (Embedded DML Precompiler): biến đổi các lệnh DML được nhúng trong một chương trình ứng dụng thành các lời gọi thủ tục chuẩn trong ngôn ngữ chủ. Trình tiền biên dịch phải trao đổi với trình biên dịch DML để sinh mã thích hợp.

- Bộ thông dịch DDL (DDL interpreter): thông dịch các lệnh DDL và ghi chúng vào một tập hợp các bảng chứa metadata.

- Engine định giá vấn tin (Query evaluation engine): Thực hiện các chỉ thị mức thấp được sinh ra bởi trình biên dịch DML.

Các thành phần quản trị lưu trữ cung cấp các giao diện giữa dữ liệu mức thấp được lưu trữ trong CSDL và các chương trình ứng dụng, các vấn tin được đệ trình cho hệ thống. Các thành phần quản trị lưu trữ gồm:

- Bộ quản trị quyền và tính toàn vẹn (Authorization and integrity manager): kiểm tra sự thoả mãn các ràng buộc toàn vẹn và kiểm tra quyền truy xuất dữ liệu của người sử dụng.

- Bộ quản trị giao dịch (Transaction manager): Đảm bảo rằng CSDL được duy trì trong trạng thái nhất quán cho dù hệ thống có sự cố và đảm bảo rằng các thực hiện giao dịch cạnh tranh tiến triển không xung đột.
- Bộ quản trị file (File manager): Quản trị cấp phát không gian trên lưu trữ đĩa và các cấu trúc dữ liệu được dùng để biểu diễn thông tin được lưu trữ trên đĩa.
- Bộ quản trị bộ đệm (Buffer manager): có trách nhiệm đem dữ liệu từ lưu trữ đĩa vào bộ nhớ chính và quyết định dữ liệu nào trữ trong bộ nhớ.

Hơn nữa, một số cấu trúc dữ liệu được cần đến như bộ phận của sự thực thi hệ thống vật lý:

- Các file dữ liệu: Lưu trữ CSDL
- Tự điển dữ liệu (Data Dictionary): lưu metadata về cấu trúc CSDL.
- Chỉ mục (Indices): cung cấp truy xuất nhanh đến các hạng mục dữ liệu chứa các giá trị tìm kiếm.
- Dữ liệu thống kê (Statistical data): lưu trữ thông tin thống kê về dữ liệu trong cơ sở dữ liệu. Thông tin này được dùng bởi bộ xử lý vấn tin để chọn những phương pháp hiệu quả thực hiện câu vấn tin.

Sơ đồ các thành phần và các nối kết giữa chúng

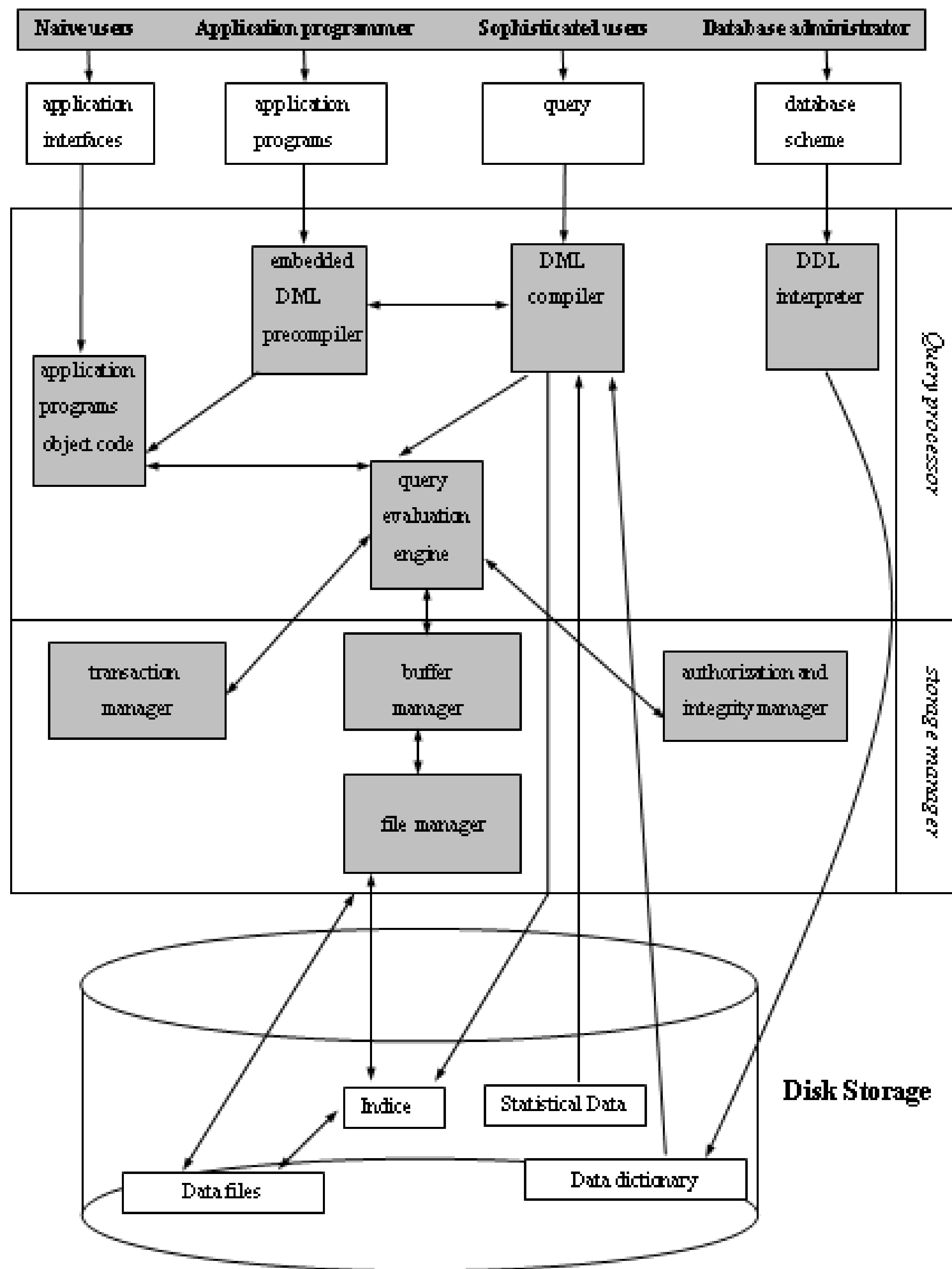

Naive users
Application programmer
Sophisticated users
Database administrator
application interfaces
application programs
query
database scheme
embedded DML precompiler
DML compiler
DDL interpreter
application programs object code
query evaluation engine
Query processor
transaction manager
buffer manager
authorization and integrity manager
file manager
storage manager
Indice
Statistical Data
Disk Storage
Data files
Data dictionary

Figure 2

KIẾN TRÚC HỆ CƠ SỞ DỮ LIỆU

Kiến trúc hệ CSDL bị ảnh hưởng nhiều bởi hệ thống máy nền. Các sắc thái của kiến trúc máy như mạng, song song và phân tán được phản ánh trong kiến trúc của hệ CSDL.

- Mạng máy tính cho phép thực hiện một số công việc trên một hệ thống các server, một số công việc trên các hệ thống client. Việc phân chia công việc này dẫn đến sự phát triển hệ CSDL client-server.

- Xử lý song song trong một hệ thống máy tính làm tăng tốc độ các hoạt động của hệ CSDL, trả lời các giao dịch nhanh hơn. Các vấn tin được xử lý theo cách khai thác tính song song. Sự cần thiết xử lý vấn tin song song này dẫn tới sự phát triển của hệ CSDL song song.

- Dữ liệu phân tán trên các site hoặc trên các bộ phận trong một cơ quan cho phép các dữ liệu thường trú tại nơi chúng được sinh ra nhưng vẫn có thể truy xuất chúng từ các site khác hay các bộ phận khác. Việc lưu nhiều bản sao của CSDL trên các site khác nhau cho phép các tổ chức lớn vẫn có thể tiếp tục hoạt động khi một hay một vài site bị sự cố. Hệ CSDL phân tán được phát triển để quản lý dữ liệu phân tán, trên phương diện địa lý hay quản trị, trải rộng trên nhiều hệ CSDL .

HỆ THỐNG TẬP TRUNG

Các hệ CSDL tập trung chạy trên máy đơn và không trao đổi với các máy khác. Các hệ thống như vậy trải từ các hệ CSDL một người sử dụng chạy trên các máy cá nhân (PC) đến các hệ CSDL hiệu năng cao chạy trên các hệ mainframe. Một hệ máy tính mục đích chung hiện đại gồm một hoặc một vài CPU và một số bộ điều khiển thiết bị được nối với nhau thông qua một bus chung, cho phép truy xuất đến bộ nhớ chia sẻ. CPU có bộ nhớ cache cục bộ lưu các bản sao của một số phần của bộ nhớ chính nhằm tăng tốc độ truy xuất dữ liệu. Mỗi bộ điều khiển thiết

bị phụ trách một kiểu thiết bị xác định. Các CPU và các bộ điều khiển thiết bị có thể thực hiện đồng thời, canh tranh truy cập bộ nhớ. Bộ nhớ cache giúp làm giảm sự tranh chấp truy xuất bộ nhớ. Ta phân biệt hai cách các máy tính được sử dụng: Hệ thống một người dùng và hệ thống nhiều người dùng. Hệ CSDL được thiết kế cho hệ thống một người dùng không hỗ trợ điều khiển cạnh tranh, chức năng phục hồi hoặc là thiếu hoặc chỉ là một sự chép dự phòng đơn giản.

HỆ THỐNG CLIENT-SERVER

Các máy tính cá nhân (PC) ngày càng trở nên mạnh hơn, nhanh hơn, và rẻ hơn. Có sự chuyển dịch trong hệ thống tập trung. Các đầu cuối (terminal) được nối với hệ thống tập trung bây giờ được thế chỗ bởi các máy tính cá nhân. Chức năng giao diện người dùng (user interface) thường được quản lý trực tiếp bởi các hệ thống tập trung nay được quản lý bởi các máy tính cá nhân. Như vậy, các hệ thống tập trung ngày nay hoạt động như các hệ thống server nó làm thoả mãn các đòi hỏi của các client. Chức năng CSDL có thể được chia thành hai phần: phần trước (front-end) và phần sau (back-end). Phần sau quản trị truy xuất cấu trúc, định giá câu vấn tin và tối ưu hoá, điều khiển sự xảy ra đồng thời và phục hồi. Phần trước của hệ CSDL gồm các công cụ như: tạo mẫu (form), các bộ soạn báo cáo (report writer), giao diện đồ hoạ người dùng (graphical user interface). Giao diện giữa phần trước và phần sau thông qua SQL hoặc một chương trình ứng dụng. Các hệ thống server có thể được phân thành các phạm trù : server giao dịch (transaction server), server dữ liệu (data server).

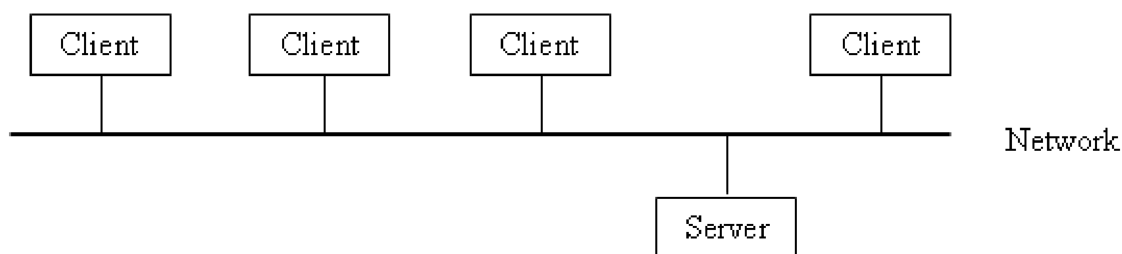


Figure 3

Hệ thống server giao dịch (transaction-server systems): còn được gọi là hệ thống server vấn tin (query-server system), cung cấp một giao diện mà các client có thể gửi đến nó các yêu cầu thực hiện một hành động. Để đáp ứng các yêu cầu, hệ thống thực hiện các hành động và gửi lại client các kết quả. Các người sử dụng có thể đặc tả các yêu cầu trong SQL hoặc trong một giao diện trình ứng dụng sử dụng một cơ chế gọi thủ tục xa (remote-procedure-call).

- Các servers giao dịch (Transaction servers): Trong các hệ thống tập trung, phần trước (front-end) và phần sau (back-end) được thực hiện trong một hệ thống. Kiến trúc server giao dịch cho phép chia chức năng giữa phần trước và phần sau. Chức năng phần trước được hỗ trợ trên các máy tính cá nhân (PC). Các PC hành động như những khách hàng của các hệ thống server nơi lưu trữ một khối lượng lớn dữ liệu và hỗ trợ các chức năng phần sau. Các clients gửi các giao dịch đến các hệ thống server tại đó các giao dịch được thực hiện và các kết quả được gửi trả lại cho các clients, người giữ trách nhiệm hiển thị dữ liệu.

ODBC (Open DataBase Connectivity) được phát triển để tạo giao diện giữa các clients và các servers. ODBC là một giao diện trình ứng dụng cho phép các clients sinh ra các lệnh SQL và gửi đến một server tại đó lệnh được thực hiện. Bất kỳ client nào sử dụng giao diện có thể nối với bất kỳ một server nào cung cấp giao diện này.

Các giao diện client-server khác ODBC cũng được sử dụng trong một số hệ thống xử lý giao dịch. Chúng được xác định bởi một giao diện lập trình ứng dụng, sử dụng nó các clients tạo ra các lời gọi thủ tục giao dịch từ xa (transactional remote procedure calls) trên server. Các lời gọi này giống như các lời gọi thủ tục gốc đối với người lập trình nhưng tất cả các lời gọi thủ tục từ xa của một client được bao trong một giao dịch ở server cuối. Như vậy nếu giao dịch bỏ dở, server có thể huỷ bỏ hiệu quả của các lời gọi thủ tục xa riêng lẻ.

Hệ thống server dữ liệu (Data-server systems): cho phép các clients trao đổi với các server bằng cách tạo ra các yêu cầu đọc hoặc cập nhật dữ liệu trong các đơn vị như file hoặc trang. Ví dụ, các file-servers cung cấp một giao diện với hệ thống file tại đó các clients có thể tạo, cập nhật, đọc hoặc xoá files. Các servers dữ liệu của cơ sở dữ liệu cung cấp nhiều chức năng hơn; chúng hỗ trợ các đơn vị dữ liệu nhỏ hơn file như trang, bộ (tuple) hoặc đối tượng. Chúng cũng cung cấp phương tiện dễ dàng để lấy chỉ mục (indexing) dữ liệu, phương tiện dễ dàng để tạo giao dịch.

- Các server dữ liệu (Data Servers): Các hệ thống server dữ liệu được sử dụng trong các mạng cục bộ, trong đó có một nối kết tốc độ cao giữa các máy clients và máy server, các máy clients có sức mạnh xử lý tương thích với máy server và các công việc phải được thực hiện là tăng cường tính toán. Trong một môi trường như vậy, có thể gửi dữ liệu đến các máy client để thực hiện tất cả các xử lý tại máy clients sau đó gửi dữ liệu trở lại đến máy server. Kiến trúc này đòi hỏi các tính năng back-end đầy đủ tại các clients. Kiến trúc server dữ liệu thường được gặp trong các hệ CSDL hướng đối tượng (Object-Oriented DataBase Systems)

Gửi trang đối lại với gửi hạng mục (Page shipping versus item shipping): Đơn vị liên lạc dữ liệu có thể là các "hạt thô" (Coarse granularity) như một trang, hay hạt min (fine granularity) như một bộ (tuple)/ đối tượng (object). Ta dùng thuật ngữ hạng mục để chỉ bộ hay đối tượng. Nếu đơn vị liên lạc là một hạng mục sẽ dẫn đến tổng chi phí truyền thông điệp tăng. Đem về hạng mục (fetching item) trước khi nó được yêu cầu, được gọi là đem về trước (Prefetching). Gửi trang có thể được xem như một dạng của đem về trước nếu một trang chứa nhiều hạng mục.

Chốt (Locking): Các chốt thường được cấp bởi server trên các hạng mục mà nó gửi cho các máy clients. Khi client giữ một chốt trên một hạng mục dữ liệu, nó có quyền “sử dụng” hạng mục dữ liệu này, hơn nữa trong khoảng thời gian client giữ chốt trên hạng mục dữ liệu không một client nào khác có thể sử dụng hạng mục dữ liệu này. Bất lợi của gửi trang là các máy client có thể được cấp các chốt "hạt quá thô" -- một chốt trên một trang ẩn chứa các chốt trên tất cả các hạng mục trong trang. Các

kỹ thuật nhằm tiết giảm chốt (lock deescalation) được đề nghị, trong đó server có thể yêu cầu các clients truyền trả lại các chốt trên các hạng mục cấp phát trước. Nếu máy client không cần hạng mục cấp phát trước, nó có thể truyền trả lại các chốt trên hạng mục cho server và các chốt này có thể được cấp phát cho các clients khác.

Trữ dữ liệu (Data caching): Dữ liệu được gửi đến một client với danh nghĩa một giao dịch có thể được trữ ở client, ngay cả khi giao dịch đã hoàn tất, nếu không gian lưu trữ có sẵn. Các giao dịch liên tiếp tại cùng một client có thể dùng dữ liệu được trữ. Tuy nhiên, sự kết dính dữ liệu là một vấn đề cần phải được xem xét: một giao dịch tìm thấy dữ liệu được trữ, nó phải chắc chắn rằng dữ liệu này là "mới nhất" vì các dữ liệu này có thể được cập nhật bởi một client khác sau khi chúng được trữ. Như vậy, vẫn phải trao đổi với server để kiểm tra tính hợp lệ của dữ liệu và để giành được một chốt trên dữ liệu.

Trữ chốt (Lock caching): Các chốt cũng có thể được trữ lại tại máy client. Nếu một hạng mục dữ liệu được tìm thấy trong cache và chốt yêu cầu cho một truy xuất đến hạng mục dữ liệu này cũng tìm thấy trong cache, thì việc truy xuất có thể tiến hành không cần một liên lạc nào với server. Tuy nhiên, server cũng phải lưu lại vết của các chốt được trữ. Nếu một client đòi hỏi một chốt từ server, server phải gọi lại tất cả các chốt xung đột trên cùng hạng mục dữ liệu từ tất cả các máy clients đã trữ các chốt.

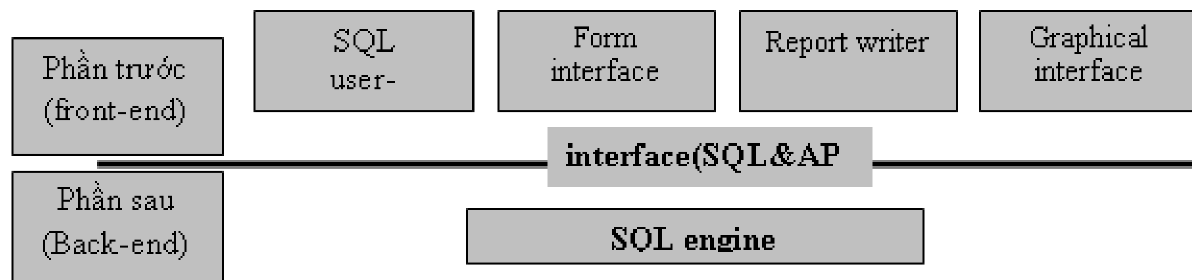


Figure 4

CÁC HỆ SONG SONG (Parallel Systems):

Các hệ song song cải tiến tốc độ xử lý và tốc độ I/O bằng cách sử dụng nhiều CPU và nhiều đĩa song song. Trong xử lý song song, nhiều hoạt động được thực hiện đồng thời. Một máy song song "hạt thô" (coarse-grain) gồm một số nhỏ các bộ xử lý mạnh. Một máy song song đồ sộ (massively parallel) hay "hạt mịn" (fine-grain) sử dụng hàng ngàn bộ xử lý nhỏ hơn. Có hai biện pháp chính để đánh giá hiệu năng của một hệ CSDL. Thứ nhất là năng lực truyền qua (throughput): số công việc có thể được hoàn tất trong một khoảng thời gian đã cho. Thứ hai là thời gian đáp ứng (response time): lượng thời gian cần thiết để hoàn thành một công việc từ lúc nó được đệ trình. Một hệ thống xử lý một lượng lớn các giao dịch nhỏ có thể cải tiến năng lực truyền qua bởi xử lý song song nhiều giao dịch. Một hệ thống xử lý các giao dịch lớn có thể cải tiến thời gian đáp ứng cũng như năng lực truyền qua bởi thực hiện song song các công việc con (subtask) của mỗi giao dịch.

- Tăng tốc độ và tăng quy mô (Speedup & Scaleup): Tăng tốc độ ám chỉ việc chạy một công việc đã cho trong thời gian ngắn hơn bằng cách tăng bậc song song. Tăng quy mô ám chỉ việc quản lý các công việc lớn bằng cách tăng bậc song song. Chúng ta hãy xét một ứng dụng CSDL chạy trên một hệ thống song song với một số processor và một số đĩa. Giả sử, chúng ta tăng kích cỡ của hệ thống bằng cách tăng số processor, đĩa, và các thành phần khác của hệ thống. Mục đích là xử lý công việc trong thời gian tỷ lệ nghịch với số processor và đĩa được cấp phát.

Giả sử, thời gian thực hiện một công việc trên một máy tính lớn là TL, thời gian thực hiện cùng công việc này trên máy tính nhỏ là TS. Tăng tốc độ nhờ song song được định nghĩa là tỷ số TS/TL, hệ thống song song được gọi là tăng tốc độ tuyến tính nếu tốc độ tăng là N khi hệ thống lớn có N lần tài nguyên (CPU, đĩa ...) lớn hơn hệ thống nhỏ. Nếu tốc độ tăng nhỏ hơn N, hệ thống được gọi là tăng tốc độ hạ tuyến tính (sublinear).

Tăng quy mô liên quan đến khả năng xử lý các công việc lớn trong trong cùng một lượng thời gian bằng cách cung cấp thêm tài nguyên. Giả sử, Q

là một công việc, QN là một công việc N lần lớn hơn Q. Giả sử thời gian thực hiện công việc Q trên một máy MS là TS và thời gian thực hiện công việc QN trên một máy song song ML, N lần lớn hơn MS, là TL. Tăng quy mô được định nghĩa là TS/TL. Hệ song song ML được gọi là tăng quy mô tuyến tính trên công việc Q nếu TS = TL. Nếu TL > TS, hệ thống được gọi là tăng quy mô hạ tuyến tính. Tăng quy mô là một độ đo (metric) quan trọng hơn trong đo lường hiệu quả của các hệ CSDL song song. Đích của song song trong các hệ CSDL là đảm bảo hệ CSDL có thể tiếp tục thực hiện ở một tốc độ chấp nhận được, ngay cả khi kích cỡ của CSDL và số giao dịch tăng lên. Tăng khả năng của hệ thống bằng cách tăng sự song song cung cấp một con đường thuận tiện hơn cho sự phát triển hơn là thay thế một hệ tập trung bởi một máy nhanh hơn. Một số nhân tố ảnh hưởng xấu đến tính hiệu quả của hoạt động song song và có thể làm giảm cả tăng tốc độ và tăng quy mô là:

- Chi phí khởi động (Startup Costs): Có một chi phí khởi động kết hợp với sự khởi động một xử lý. Trong một hoạt động song song gồm hàng ngàn xử lý, thời gian khởi động (Startup time) có thể làm lu mờ thời gian xử lý hiện tại, ảnh hưởng bất lợi tới tăng tốc độ.
- Sự giao thoa (Interference): Các xử lý thực hiện trong một hệ song song thường truy nhập đến các tài nguyên chia sẻ, một sự giao thoa của mỗi xử lý mới khi nó cạnh tranh với các xử lý đang tồn tại trên các tài nguyên bị chiếm như bus hệ thống, đĩa chia sẻ, thậm chí cả đồng hồ. Hiện tượng này ảnh hướng đến cả tăng tốc độ lẫn tăng quy mô.

- Sự lệch (Skew): Bằng cách chia một công việc thành các bước song song, ta làm giảm kích cỡ của bước trung bình. Tuy nhiên, thời gian phục vụ cho bước chậm nhất sẽ xác định thời gian phục vụ cho toàn bộ công việc. Thường khó có thể chia một công việc thành các phần cùng kích cỡ, như vậy cách mà kích cỡ được phân phối là bị lệch. Ví dụ: một công việc có kích cỡ 100 được chia thành 10 phần và sự phân chia này bị lệch, có thể có một số phần có kích cỡ nhỏ hơn 10 và một số nhiệm vụ có kích cỡ lớn hơn 10, giả sử trong đó có phần kích cỡ 20, độ tăng tốc nhận được bởi chạy các công việc song song chỉ là 5, thay vì 10.

- Các mạng hợp nhất (Interconnection Network): Các hệ thống song song gồm một tập hợp các thành phần (Processors, memory và các đĩa) có thể liên lạc với nhau thông qua một mạng hợp nhất. Các ví dụ về các mạng hợp nhất là:

- Bus: Toàn bộ các thành phần hệ thống có thể gửi và nhận dữ liệu qua một bus liên lạc. Các kiến trúc bus làm việc tốt với một số nhỏ các processor. Tuy nhiên, chúng không có cùng quy mô khi tăng sự song song vì bus chỉ điều khiển liên lạc từ chỉ một thành phần tại một thời điểm.

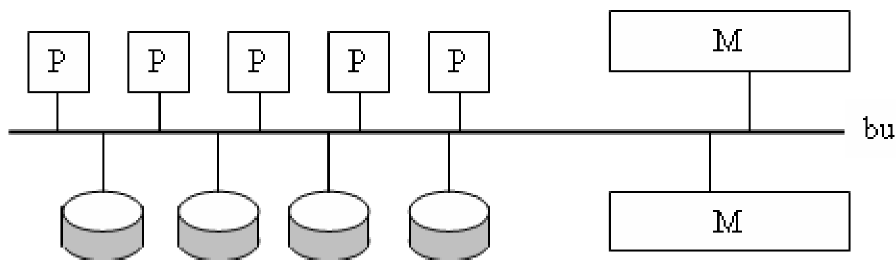
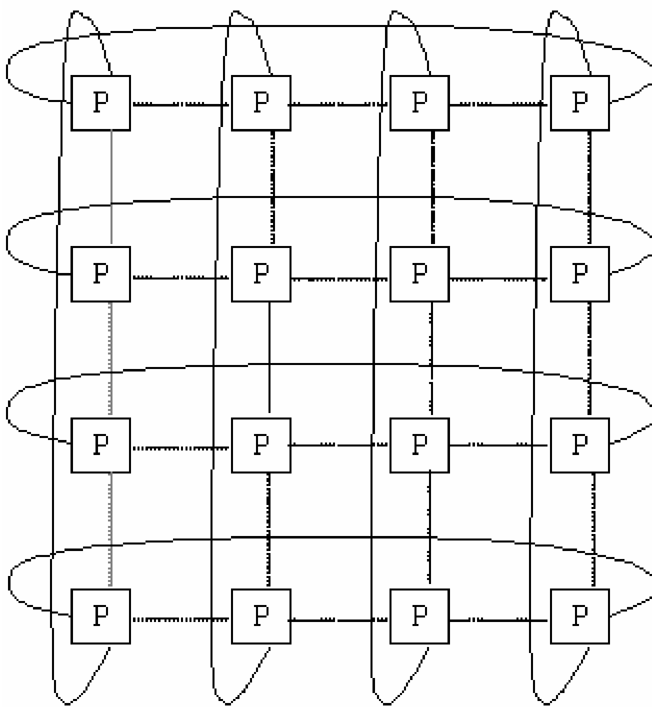


Figure 5

- Lưới (Mesh): Các thành phần được sắp xếp như các nút của một lưới, mỗi thành phần được nối với tất cả các thành phần kề với nó trong lưới. Trong một lưới hai chiều mỗi nút được nối với 4 nút kề.



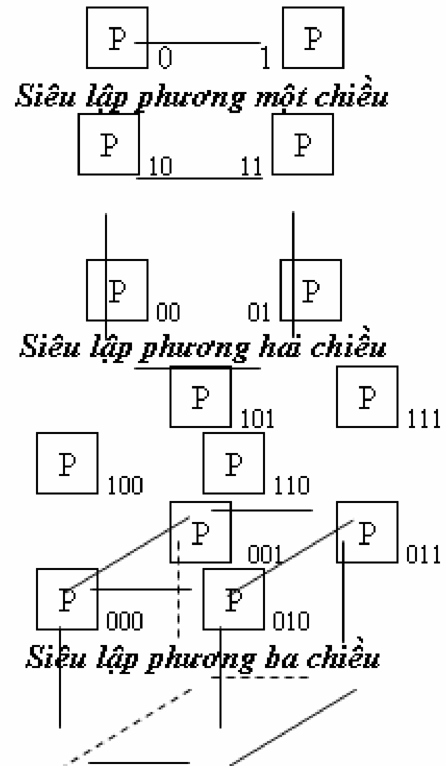
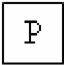


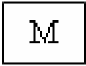
Figure 6

- Siêu lập phương (Hypercube): Các thành phần được đánh số theo nhị phân, một thành phần được nối với thành phần khác nếu biểu diễn nhị nhân số của chúng sai khác nhau đúng một bit. Như vậy, mỗi một thành phần trong n thành phần được nối với log(n) thành phần khác. Có thể kiểm nghiệm được rằng trong một sự hợp nhất siêu lập phương một thông điệp từ một thành phần đến một thành phần khác đi quá nhiều nhất log(n) nối kết, còn trong lưới là $\sqrt{n} - 1$
- Các kiến trúc cơ sở dữ liệu song song: Có một vài mô hình kiến trúc cho các máy song song:

(Ký hiệu:



= processor,



= Memory,



= đĩa)

- Bộ nhớ chia sẻ (Shared memory): Tất cả các processor chia sẻ một bộ nhớ chung. Trong mô hình này các processor và các đĩa truy xuất một bộ nhớ chung, thường thông qua một bus hoặc một mạng hợp nhất. Thuận lợi của chia sẻ bộ nhớ là liên lạc giữa các processor là cực kỳ hiệu quả: dữ liệu trong bộ nhớ chia sẻ có thể được truy xuất bởi bất kỳ processor nào mà không phải di chuyển bởi phần mềm. Một processor có thể gửi thông điệp cho một processor khác bằng cách viết vào bộ nhớ chia sẻ, cách liên lạc này nhanh hơn nhiều so với các liên lạc khác. Tuy nhiên, các máy bộ nhớ chia sẻ không thể hỗ trợ nhiều hơn 64 processor vì nếu nhiều hơn, bus hoặc mạng hợp nhất sẽ trở nên dễ bị nghẽn (bottle-neck). Kiến trúc bộ nhớ chia sẻ thường có những cache lớn cho mỗi processor, như vậy việc tham khảo bộ nhớ chia sẻ có thể tránh được mỗi khi có thể.

[missing_resource: graphics10.wmf]

Figure 7

- Đĩa chia sẻ (Shared disk): Tất cả các processor chia sẻ đĩa chung. Mô hình này còn được gọi là cụm (cluster). Trong mô hình này tất cả các processor có thể truy xuất trực tiếp đến tất cả các đĩa thông qua một mạng hợp nhất, nhưng mỗi processor có bộ nhớ riêng.

[missing_resource: graphics11.wmf]

Figure 8

Kiến trúc này có các điểm thuận lợi là: thứ nhất, bus bộ nhớ không bị bottle-neck, thứ hai, cho một phương pháp rẻ để cung cấp một mức độ lượng thứ lỗi--một processor bị hỏng hóc, các processor khác có thể tiếp tục công việc của nó. Ta có thể tạo ra hệ thống con các đĩa tự lượng thứ lỗi bằng cách sử dụng kiến trúc RAID (được trình bày sau này). Vấn đề chính của chia sẻ đĩa là sự hợp nhất các hệ thống con các đĩa trở nên bottle-neck, đặc biệt trong tình huống CSDL truy xuất đĩa nhiều. So sánh với bộ nhớ chia sẻ, chia sẻ đĩa có thể hỗ trợ một số lượng processor lớn hơn, nhưng việc liên lạc giữa các processor chậm hơn.

- Không chia sẻ (Shared nothing): Các processor không chia sẻ bộ nhớ chung, cũng không chia sẻ đĩa chung. Trong hệ thống này mỗi nút của máy có một processor, bộ nhớ và một vài đĩa.

[missing_resource: graphics12.wmf]

Figure 9

Các processor ở mỗi nút có thể liên lạc với các processor khác qua mạng hợp nhất tốc độ cao. Chức năng của một nút, như server, dữ liệu được chứa trên các đĩa của nó. Mô hình không chia sẻ gì chỉ có vấn đề về việc truy xuất các đĩa không cục bộ và việc truyền các quan hệ kết quả qua mạng. Hơn nữa, đối với các hệ thống không chia sẻ gì, các mạng hợp nhất thường được thiết kế để có thể tăng quy mô, sao cho khả năng truyền của chúng tăng khi các nút mới được thêm vào.

- Phân cấp (hierarchical): Mô hình này là một sự lai kiểu của các kiến trúc trước.

[missing_resource: graphics13.wmf]

Figure 10

Kiến trúc này tổ hợp các đặc trưng của các kiến trúc chia sẻ bộ nhớ, chia sẻ đĩa và không chia sẻ gì. Ở mức cao nhất, hệ thống bao gồm những nút được nối bởi mạng hợp nhất và không chia sẻ đĩa cũng như bộ nhớ với nút khác. Như vậy, mức cao nhất là kiến trúc không chia sẻ gì. Mỗi nút của hệ thống có thể là hệ thống chia sẻ bộ nhớ với một vài processor. Kế tiếp, mỗi nút có thể là một hệ thống chia sẻ đĩa. Mỗi một hệ thống chia sẻ đĩa lại có thể là một hệ thống chia sẻ bộ nhớ... Như vậy, hệ thống có thể được xây dựng như một sự phân cấp.

CÁC HỆ THỐNG PHÂN TÁN (Distributed Systems):

Trong một hệ thống CSDL phân tán, CSDL được lưu trữ trên một vài máy tính. Các máy tính trong một hệ thống phân tán liên lạc với một máy khác qua nhiều dạng phương tiện liên lạc khác nhau: mạng tốc độ cao, đường điện thoại... Chúng không chia sẻ bộ nhớ cũng như đĩa. Các máy tính trong hệ thống phân tán có thể rất đa dạng về kích cỡ cũng như chức năng: từ các workstation đến các mainframe. Các máy tính trong hệ thống phân tán được tham chiếu bởi một số các tên khác nhau: site , node -- phụ thuộc vào ngữ cảnh mà máy được đề cập. Ta sẽ sử dụng thuật ngữ site để nhấn mạnh sự phân tán vật lý của các hệ thống này.

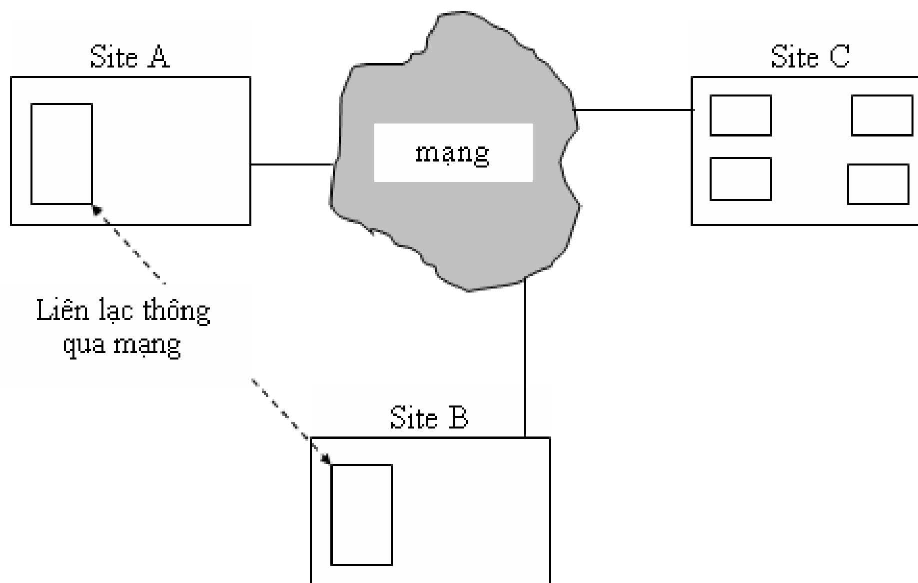


Figure 11

Sự sai khác chính giữa CSDL song song không chia sẻ gì và hệ thống phân tán là CSDL phân tán được tách biệt về mặt địa lý, được quản trị tách biệt và có một sự hợp nhất chậm. Hơn nữa, trong hệ thống phân tán người ta phân biệt giữa các giao dịch cục bộ (local) và toàn thể (global). Giao dịch cục bộ là một giao dịch truy xuất dữ liệu trong một site tại đó giao dịch đã được khởi xướng. Giao dịch toàn thể là một giao dịch mà nó hoặc truy xuất dữ liệu trong một site từ một site khác tại đó nó được khởi xướng hoặc truy xuất dữ liệu trong một vài site khác nhau.

BÀI TẬP CHƯƠNG I

I.1Bốn điểm khác nhau chính giữa một hệ thống xử lý file và một hệ quản trị CSDL là gì ?

I.2Giải thích sự khác nhau giữa độc lập dữ liệu vật lý và độc lập dữ liệu logic

I.3Liệt kê năm nhiệm vụ của nhà quản trị CSDL. Đối với mỗi nhiệm vụ, giải tích rõ những vấn đề nảy sinh nếu nhiệm vụ đó không được hoàn thành.

I.4Năm chức năng chính của nhà quản trị CSDL là gì ?

I.5Sử dụng một mảng hai chiều kích cỡ n x m như một ví dụ để minh hoạ sự khác nhau:

1. giữa ba mức trừu tượng dữ liệu
2. giữa sơ đồ và thể hiện

I.6Trong các hệ thống client-server tiêu biểu, máy server thường mạnh hơn máy client rất nhiều: Có bộ xử lý nhanh hơn thậm chí có thể có nhiều bộ xử lý, có bộ nhớ lớn hơn, có dung lượng đĩa lớn hơn. Ta xét một kịch bản trong đó các máy client và các máy server là mạnh như nhau. Xây dựng một hệ thống client-server theo kịch bản như vậy có ưu nhược điểm gì ? Kịch bản nào phù hợp hơn với kiến trúc server dữ liệu ?

I.7Giả sử một giao dịch được viết trong C với SQL nhúng, và khoảng 80% thời gian được dùng cho code SQL, 20% còn lại cho code C. Nếu song song được dùng chỉ cho code SQL, Tăng tốc độ có thể đạt tới bao nhiêu ? Giải thích.

I.8Những nhân tố nào chống lại việc tăng quy mô tuyến tính trong một hệ thống xử lý giao dịch ? Nhân tố nào là quan trọng nhất trong mỗi một kiến trúc sau: bộ nhớ chia sẻ, đĩa chia sẻ, không chia sẻ gì ?

I.9Xét một mạng dựa trên đường điện thoại quay số tự động, trong đó các site liên lạc theo định kỳ, ví dụ hàng đêm. Các mạng như vậy thường được cấu hình với một site server và nhiều site client. Các site client chỉ nói với server và trao đổi dữ liệu với client khác bởi lưu dữ liệu tại server và lấy dữ liệu được lưu trên server bởi client khác. Ưu, nhược điểm của kiến trúc như vậy là gì ?

Sql

MỤC ĐÍCH Giới thiệu một hệ CSDL chuẩn, SQL, các thành phần cơ bản của của nó. YÊU CẦU Hiểu các thành phần cơ bản của SQL-92 Hiểu và vận dụng phương pháp "dịch" từ câu vấn tin trong ngôn ngữ tự nhiên sang ngôn ngữ SQL và ngược lại Hiểu và vận dụng cách thêm (xen), xóa dữ liệu SQL là ngôn ngữ CSDL quan hệ chuẩn, gốc của nó được gọi là Sequel. SQL là viết tắt của Structured Query Language. Có nhiều phiên bản của SQL. Phiên bản được trình bày trong giáo trình này là phiên bản chuẩn SQL-92. SQL có các phần sau: • Ngôn ngữ định nghĩa dữ liệu (DDL). DDL của SQL cung cấp các lệnh để định nghĩa các sơ đồ quan hệ, xoá các quan hệ, tạo các chỉ mục, sủa đổi các sơ đồ quan hệ • Ngôn ngữ thao tác dữ liệu tương tác (Interactive DML). IDML bao gồm một ngôn ngữ dựa trên cả đại số quan hệ lẫn phép tính quan hệ bộ. Nó bao hàm các lệnh xen các bộ, xoá các bộ, sửa đổi các bộ trong CSDL • Ngôn ngữ thao tác dữ liệu nhúng (Embedded DML). Dạng SQL nhúng được thiết kế cho việc sử dụng bên trong các ngôn ngữ lập trình mục đích chung (genaral-purpose programming languages) như PL/I, Cobol, Pascal, Fortran, C. • Đinh nghĩa view. DDL SQL cũng bao hàm các lệnh để định nghĩa các view. • Cấp quyền (Authorization). DDL SQL bao hàm cả các lệnh để xác định các quyền truy xuất đến các quan hệ và các view • Tính toàn vẹn (Integrity). DDL SQL chứa các lệnh để xác định các ràng buộc toàn vẹn mà dữ liệu được lưu trữ trong CSDL phải thoả. • Điều khiển giao dịch. SQL chứa các lệnh để xác định bắt đầu và kết thúc giao dịch, cũng cho phép chốt tường minh dữ liệu để điều khiển cạnh tranh Các ví dụ minh hoạ cho các câu lệnh SQL được thực hiện trên các sơ đồ quan hệ sau: • Branch_schema = (Branch_name, Branch_city, Assets): Sơ đồ quan hệ chi nhánh nhà băng gồm các thuộc tính Tên chi nhánh (Branch_name), Thành phố (Branch_city), tài sản (Assets) • Customer_schema = (Customer_name, Customer_street, Customer_city): Sơ đồ quan hệ Khách hàng gồm các thuộc tính Tên khách hàng (Customer_name), phố (Customer_street), thành phố (Customer_city) • Loan_schema = (Branch_name, loan_number, amount): Sơ đồ quan hệ cho vay gồm các thuộc tính Tên chi nhánh, số cho vay (Loan_number), số lượng (Amount) • Borrower_schema = (Customer_name, loan_number): Sơ đồ quan hệ người mượn gồm các thuộc tính Tên khách hàng, số cho vay • Account_schema = (Branch_name, account_number, balance): Sơ đồ

quan hệ tài khoản gồm các thuộc tính Tên chi nhánh, số tài khoản (Account_number), số cân đối (Balance: dư nợ/có) • Depositor_schema = (Customer_name, account_number): Sơ đồ người gửi gồm các thuộc tính Tên khách hàng, số tài khoản Cấu trúc cơ sở của một biểu thức SQL gồm ba mệnh đề: SELECT, FROM và WHERE • Mệnh đề SELECT tương ứng với phép chiếu trong đại số quan hệ, nó được sử dụng để liệt kê các thuộc tính mong muốn trong kết quả của một câu vấn tin • Mệnh đề FROM tương ứng với phép tích Đề các , nó nó liệt kê các quan hệ được quét qua trong sự định trị biểu thức • Mệnh đề WHERE tương ứng với vị từ chọn lọc, nó gồm một vị từ chứa các thuộc tính của các quan hệ xuất hiện sau FROM Một câu vấn tin kiểu mẫu có dạng: SELECT A1, A2, ..., Ak FROM R1, R2, ..., Rm WHERE P trong đó Ai là các thuộc tính (Attribute), Rj là các quan hệ (Relation) và P là một vị từ (Predicate). Nếu thiếu WHERE vị từ P là TRUE. Kết quả của một câu vấn tin SQL là một quan hệ.

MỆNH ĐỀ SELECT

Ta tìm hiểu mệnh đề SELECT bằng cách xét một vài ví dụ:

"Tìm kiếm tất cả các tên các chi nhánh trong quan hệ cho vay (loan)":

SELECT Branch_name

FROM Loan;

Kết quả là một quan hệ gồm một thuộc tính Tên chi nhánh (Branch_name)

Nếu muốn quan hệ kết quả không chứa các tên chi nhánh trùng nhau:

SELECT DISTINCT Branch_name

FROME Loan;

Từ khoá ALL được sử dụng để xác định tường minh rằng các giá trị trùng không bị xoá và nó là mặc nhiên của mệnh đề SELECT.

Ký tự * được dùng để chỉ tất cả các thuộc tính:

SELECT *

FROM Loan;

Sau mệnh đề SELECT cho phép các biểu thức số học gồm các phép toán +, -, *, / trên các hằng hoặc các thuộc tính:

SELECT Branch_name, Loan_number, amount * 100

FROM Loan;

MỆNH ĐỀ WHERE

“Tìm tất cả các số cho vay ở chi nhánh tên Perryridge với số lượng vay lớn hơn1200$"

SELECT Loan_number

FROM Loan

WHERE Branch_name = ‘Perryridge’ AND Amount > 1200;

SQL sử dụng các phép nối logic: NOT, AND, OR. Các toán hạng của các phép nối logic có thể là các biểu thức chứa các toán tử so sánh =, >=, <>, <, <=.

Toán tử so sánh BETWEEN được dùng để chỉ các giá trị nằm trong một khoảng:

SELECT Loan_number

FROM Loan

WHERE Amount BETWEEN 50000 AND 100000;

SELECT Loan_number

FROM Loan

WHERE Amount >= 50000 AND Amount <= 100000;

Ta cũng có thể sử dụng toán tử NOT BETWEEN.

MỆNH ĐỀ FROM

"Trong tất cả các khách hàng có vay ngân hàng tìm tên và số cho vay của họ"

SELECT DISTINCT Customer_name, Borrower.Loan_number

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number;

SQL sử dụng cách viết <tên quan hệ >.< tên thuộc tính > để che dấu tính lập lờ trong trường hợp tên thuộc tính trong các sơ đồ quan hệ trùng nhau.

"Tìm các tên và số cho vay của tất cả các khách hàng có vay ở chi nhánh Perryridge"

SELECT Customer_name, Borrower.Loan_number

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number AND

Branch_name =’Perryridge’;

CÁC PHÉP ĐỔI TÊN

SQL cung cấp một cơ chế đổi tên cả tên quan hệ lẫn tên thuộc tính bằng mệnh đề dạng:

< tên cũ > AS < tên mới >

mà nó có thể xuất hiện trong cả mệnh đề SELECT lẫn FROM

SELECT DISTINCT Customer_name, Borrower.Loan_number

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number AND

Branch_name =’Perryridge’;

Kết quả của câu vấn tin này là một quan hệ hai thuộc tính: Customer_name, Loan_number

Đổi tên thuộc tính của quan hệ kết quả:

SELECT Customer_name, Borrower.Loan_number AS Loan_Id

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number AND

Branch_name =’Perryridge’;

CÁC BIẾN BỘ (Tuple Variables)

Các biến bộ được định nghĩa trong mệnh đề FROM thông qua sử dụng mệnh đề AS:

SELECT DISTINCT Customer_name, T.Loan_number

FROM Borrower AS T, Loan AS S

WHERE T.Loan_number = S.Loan_number AND

Branch_name =’Perryridge’;

“Tìm các tên của tất cả các chi nhánh có tài sản lớn hơn ít nhất một chi nhánh ở Brooklyn“

SELECT DISTINCT T.branch_name

FROM Branch AS T, Banch AS S

WHERE T.assets > S.assets AND S.Branch_City = ‘Brooklyn’

SQL92 cho phép sử dụng các viết (v1, v2, ..., vn) để ký hiệu một n-bộ với các giá trị v1, v2, ..., vn. Các toán tử so sánh có thể được sử dụng trên các n-bộ và theo thứ tự tự điển. Ví dụ (a1, b1) <= (a2, b2) là đúng nếu (a1 < b1) OR ((a1 = b1) AND (a2 < b2)).

CÁC PHÉP TOÁN TRÊN CHUỖI

Các phép toán thường được dùng nhất trên các chuỗi là phép đối chiếu mẫu sử dụng toán tử LIKE. Ta mô tả các mẫu dùng hai ký tự đặc biệt:

- ký tự phần trăm (%): ký tự % tương xứng với chuỗi con bất kỳ
- ký tự gạch nối (_): ký tự gạch nối tương xứng với ký tự bất kỳ.

- ‘Perry%’ tương xứng với bất kỳ chuỗi nào bắt đầu bởi ‘Perry’
- ‘%idge%’ tương xứng với bất kỳ chuỗi nào chứa ‘idge’ như chuỗi con
- ‘___’ tương xứng với chuỗi bất kỳ có đúng ba ký tự
- ‘___%’ tương xứng với chuỗi bất kỳ có ít nhất ba ký tự

"Tìm tên của tất cả các khách hàng tên phố của họ chứa chuỗi con ‘Main’

SELECT Customer_name

FROM Customer

WHERE Customer_street LIKE ‘%Main%’

Nếu trong chuỗi mẫu có chứa các ký tự % _ \ , để tránh nhầm lẫn ký tự với"dấu hiệu thay thế", SQL sử dụng cách viết: ký tự escape (\) đứng ngay trước ký tự"đặc biệt". Ví dụ nếu chuỗi mẫu là ab%cd được viết là

‘ab\%cd’, chuỗi mẫu là ab_cde được viết là ‘ab_cde’, chuỗi mẫu là ab\cd được viết là ‘ab\cd’

SQL cho phép đối chiếu không tương xứng bằng cách sử dụng NOT LIKE

SQL cũng cho phép các hàm trên chuỗi: nối hai chuỗi (|), trích ra một chuỗi con, tìm độ dài chuỗi, biến đổi một chuỗi chữ thường sang chuỗi chữ hoa và ngược lại ...

THỨ TỰ TRÌNH BÀY CÁC BỘ (dòng)

Mệnh đề ORDER BY tạo ra sự trình bày các dòng kết quả của một câu vấn tin theo một trình tự. Để liết kê theo thứ tự alphabet tất cả các khách hàng có vay ở chi nhánh Perryridge:

SELECT DISTINCT Customer_name

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number AND

Branch_name = ‘Perryridge’

ORDER BY Customer_name;

Mặc nhiên, mệnh đề ORDER BY liệt kê theo thứ tự tăng, tuy nhiên ta có thể làm liệt kê theo thứ tự giảm/tăng bằng cách chỉ rõ bởi từ khoá DESC/ ASC

SELECT *

FROMLoan

ORDER BY Amount DESC, Loan_number ASC;

CÁC PHÉP TOÁN TẬP HỢP

SQL92 có các phép toán UNION, INTERSECT, EXCEPT chúng hoạt động giống như các phép toán hợp, giao, hiệu trong đại số quan hệ. Các quan hệ tham gia vào các phép toán này phải tương thích (có cùng tập các thuộc tính).

- Phép toán UNION

“tìm kiếm tất cả các khách hàng có vay, có tài khoản hoặc cả hai ở ngân hàng”

(SELECT Customer_name

FROM Depositor)

UNION

(SELECT Customer_name

FROM Borrower);

Phép toán hợp UNION tự động loại bỏ các bộ trùng, nếu ta muốn giữ lại các bộ trùng ta phải sử dụng UNION ALL

(SELECT Customer_name

FROM Depositor)

UNION ALL

(SELECT Customer_name

FROM Borrower);

- Phép toán INTERSECT

“tìm kiếm tất cả các khách hàng có vay và cả một tài khoản tại ngân hàng”

(SELECT DISTINCT Customer_name

FROM Depositor)

INTERSECT

(SELECT DISTINCT Customer_name

FROM Borrower);

Phép toán INTERESCT tự động loại bỏ các bộ trùng, Để giữ lại các bộ trùng ta sử dụng INTERSECT ALL

(SELECT Customer_name

FROM Depositor)

INTERSECT ALL

(SELECT Customer_name FROM Borrower);

- Phép toán EXCEPT

“Tìm kiếm tất cả các khách hàng có tài khoản nhưng không có vay tại ngân hàng”

(SELECT Customer_name

FROM Depositor)

EXCEPT

(SELECT Customer_name

FROM Borrower);

EXCEPT tự động loại bỏ các bộ trùng, nếu muốn giữ lại các bộ trùng phải dùng EXCEPT ALL

(SELECT Customer_name

FROM Depositor)

EXCEPT ALL

(SELECT Customer_name

FROM Borrower);

CÁC HÀM TÍNH GỘP

SQL có các hàm tính gộp (aggregate functions):

- Tính trung bình (Average): AVG()
- Tính min : MIN()
- Tính max:MAX()
- Tính tổng:SUM()
- Đếm:COUNT()

Đối số của các hàm AVG và SUM phải là kiểu dữ liệu số

"Tìm số cân đối tài khoản trung bình tại chi nhánh Perryridge”

SELECT AGV(balace)

FROM Account

WHERE Branch_name = ‘Perryridge’;

SQL sử dụng mệnh để GROUP BY vào mục đích nhóm các bộ có cùng giá trị trên các thuộc tính nào đó

"Tìm số cân đối tài khoản trung bình tại mỗi chi nhánh ngân hàng”

SELECT Branch_name, AVG(balance)

FROM Account

GROUP BY Branch_name;

“Tìm số các người gửi tiền đối với mỗi chi nhánh ngân hàng”

SELECT Branch_name, COUNT(DISTINCT Customer_name)

FROM Depositor, Account

WHERE Depositor.Account_number = Account.Acount_number

GROUP BY Branch_name

Giả sử ta muốn liệt kê các chi nhánh ngân hàng có số cân đối trung bình lớn hơn 1200$. Điều kiện này không áp dụng trên từng bộ, nó áp dụng trên từng nhóm. Để thực hiện được điều này ta sử dụng mệnh đề HAVING của SQL

SELECT Branch_name, AVG(balance)

FROM Account

GROUP BY Branch_name

HAVING AGV(Balance) > 1200$;

Vị từ trong mệnh đề HAVING được áp dụng sau khi tạo nhóm, như vậy hàm AVG có thể được sử dụng

“Tìm số cân đối đối với tất cả các tài khoản”

SELECT AVG(Balance) FROM Account;

“Đếm số bộ trong quan hệ Customer”

SELECT Count(*)FROM Customer;

SQL không cho phép sử dụng DISTINCT với COUNT(*), nhưng cho phép sử dụng DISTINCT với MIN và MAX.

Nếu WHERE và HAVING có trong cùng một câu vấn tin, vị từ sau WHERE được áp dụng trước. Các bộ thoả mãn vị từ WHERE được xếp

vào trong nhóm bởi GROUP BY, mệnh đề HAVING (nếu có) khi đó được áp dụng trên mỗi nhóm. Các nhóm không thoả mãn mệnh đề HAVING sẽ bị xoá bỏ.

“Tìm số cân đối trung bình đối với mỗi khách hàng sống ở Harrison và có ít nhất ba tài khoản”

SELECT Depositor.Customer_name, AVG(Balance)

FROM Depositor, Account, Customer

WHERE Depositor.Account_number = Account.Account_number AND

Depositor.Customer_name = Customer.Customer_name AND

Customer.city =’Harrison’

GROUP BY Depositor.Customer_name

HAVING COUNT(DISTINT Depositor.Account_number) >= 3;

CÁC GIÁ TRỊ NULL

SQL cho phép sử dụng các giá trị null để chỉ sự vắng mặt thông tin tạm thời về giá trị của một thuộc tính. Ta có thể sử dụng từ khoá đặc biệt null trong vị từ để thử một giá trị null.

"Tìm tìm tất cả các số vay trong quan hệ Loan với giá trị Amount là null"

SELECT Loan_number

FROM Loan

WHERE Amount is null

Vị từ not null thử các giá trị không rỗng

Sử dụng giá trị null trong các biểu thức số học và các biểu thức so sánh gây ra một số phiền phức. Kết quả của một biểu thức số học là null nếu một giá trị input bất kỳ là null. Kết quả của một biểu thức so sánh chứa một giá trị null có thể được xem là false. SQL92 xử lý kết quả của một phép so sánh như vậy như là một giá trị unknown, là một giá trị không là true mà cũng không là false. SQL92 cũng cho phép thử kết quả của một phép so sánh là unknown hay không. Tuy nhiên, trong hầu khắp các trường hợp, unknown được xử lý hoàn toàn giống như false.

Sự tồn tại của các giá trị null cũng làm phức tạp việc sử lý các toán tử tính gộp. Giả sử một vài bộ trong quan hệ Loan có các giá trị null trên trường Amount. Ta xét câu vấn tin sau:

SELECT SUM(Amount)

FROM LOAN

Các giá trị được lấy tổng trong câu vấn tin bao hàm cả các trị null. Thay vì tổng là null, SQL chuẩn thực hiện phép tính tổng bằng cách bỏ qua các giá trị input là null.

Nói chung, các hàm tính gộp tuân theo các quy tắc sau khi xử lý các giá trị null: Tất cả các hàm tính gộp ngoại trừ COUNT(*) bỏ qua các giá trị input null. Khi các giá trị nul bị bỏ qua, tập các giá trị input có thể là rõng. COUNT() của một tập rỗng được định nghĩa là 0. Tất cả các hàm tính gộp khác trả lại giá trị null khi áp dụng trên tập hợp input rỗng.

CÁC CÂU VẤN TIN CON LỒNG NHAU (Nested Subqueries)

SQL cung cấp một cơ chế lòng nhau của các câu vấn tin con. Một câu vấn tin con là một biểu thức SELECT-FROM-WHERE được lồng trong một caau vấn tin khác. Các câu vấn tin con thường được sử dụng để thử quan hệ thành viên tập hợp, so sánh tập hợp và bản số tập hợp.

QUAN HỆ THÀNH VIÊN TẬP HỢP (Set relationship)

SQL đưa vào các phép tính quan hệ các phép toán cho phép thử các bộ có thuộc một quan hệ nào đó hay không. Liên từ IN thử quan hệ thành viên này. Liên từ NOT IN thử quan hệ không là thành viên.

"Tìm tất cả các khách hàng có cả vay lẫn một tài khoản tại ngân hàng"

Ta đã sử dụng INTERSECTION để viết câu vấn tin này. Ta có thể viết câu vấn tin này bằng các sử dụng IN như sau:

SELECT DISTINCT Customer_name

FROM Borrower

WHERE Customer_name IN (SELECT Customer_name

FROM Depositor)

Ví dụ này thử quan hệ thành viên trong một quan hệ một thuộc tính. SQL92 cho phép thử quan hệ thành viên trên một quan hệ bất kỳ.

"Tìm tất cả các khách hàng có cả vay lãn một tài khoản ở chi nhánh Perryridge"

Ta có thể viết câu truy vấn như sau:

SELECT DISTINCT Customer_name

FROM Borrower, Loan

WHERE Borrower. Loan_number = Loan.Loan_number AND

Branch_name = 'Perryridge'AND

(Branch_name. Customer_name IN

(SELECT Branch_name, Customer_name

FROM Depositor, Account

WHEREDepositor.Account_number= Account.Account_number)

"Tìm tất cả các khách hàng có vay ngân hàng nhưng không có tài khoản tại ngân hàng"

SELECT DISTINCT Customer_name

FROM borrower

WHERE Customer_name NOT IN (SELECT Customer_name

FROM Depositor)

Các phép toán IN và NOT IN cũng có thể được sử dụng trên các tập hợp liệt kê:

SELECT DISTINCT Customer_name

FROM borrower

WHERE Customer_name NOT IN ('Smith', 'Jone')

SO SÁNH TẬP HỢP (Set Comparision)

"Tìm tên của tất cả các chi nhánh có tài sản lớn hơn ít nhất một chi nhánh đóng tại Brooklyn"

SELECT DISTINCT Branch_name

FROM Branch AS T, Branch AS S

WHERE T.assets > S.assets AND S.branch_city = 'Brooklyn'

Ta có thể viết lại câu vấn tin này bằng cách sử dụng mệnh đề"lớn hơn ít nhất một"trong SQL

- SOME :

SELECT Branch_name

FROM Branch

WHERE Assets > SOME (SELECT Assets

FROM Branch

WHERE Branch_city ='Brooklyn')

Câu vấn tin con

(SELECT Assets

FROM Branch

WHERE Branch_city ='Brooklyn')

sinh ra tập tất cả các Assets của tất cả các chi nhánh đóng tại Brooklyn. So sánh > SOME trong mệnh đề WHERE nhận giá trị đúng nếu giá trị Assets của bộ được xét lớn hơn ít nhất một trong các giá trị của tập hợp này.

SQL cũng có cho phép các so sánh < SOME, >= SOME, <= SOME, = SOME, <> SOME

- ALL

"Tìm tất cả các tên của các chi nhánh có tài sản lớn hơn tài sản của bất kỳ chi nhánh nào đóng tại Brooklyn"

SELECT Branch_name

FROM Branch

WHERE Assets > ALL (SELECT Assets

FROM Branch

WHERE Branch_citty = 'Brooklyn')

SQL cũng cho phép các phép so sánh: < ALL, <= ALL, > ALL, >= ALL, = ALL, <> ALL.

"Tìm chi nhánh có số cân đối trung bình lớn nhất"

SQL không cho phép hợp thành các hàm tính gộp, như vậy MAX(AVG (...)) là không được phép. Do vậy, ta phải sử dụng câu vấn tin con như sau:

SELECT Branch_name

FROM Account

GROUP BY Branch_name

HAVING AVG (Balance) >= ALL (SELECT AVG (balance)

FROM Account

GROUP BY Branch_name)

THỬ CÁC QUAN HỆ RỖNG

"tìm tất cả các khách hàng có cả vay lẫn tài khoản ở ngân hàng"

SELECT Customer_name

FROM Borrower

WHERE EXISTS (SELECT *

FROM Depositor

WHERE Depositor.Customer_name = Borrower.Customer_name)

Cấu trúc EXISTS trả lại giá trị true nếu quan hệ kết quả của câu vấn tin con không rỗng. SQL cũng cho phép sử dụng cấu trúc NOT EXISTS để kiểm tra tính không rỗng của một quan hệ.

"Tìm tất cả các khách hàng có tài khoản tại mỗi chi nhánh đóng tại Brooklyn"

SELECT DISTINCT S.Customer_name

FROM Depositor AS S

WHERE NOT EXISTS((SELECT Branch_name

FROM Branch

WHERE Branch_city = 'Brooklyn')

EXCEPT

(SELECT R.branch_name

FROM Depositor AS T, Account AS R

WHERE T.Acoount_number = R.Account_number

AND S.Customer_name = T.Customer_name))

THỬ KHÔNG CÓ CÁC BỘ TRÙNG

SQL đưa vào cấu trúc UNIQUE để kiểm tra việc có bộ trùng trong quan hệ kết quả của một câu vấn tin con.

"Tìm tất cả khách hàng chỉ có một tài khoản ở chi nhánh Perryridge"

SELECT T.Customer_name

FROM Depositor AS T

WHERE UNIQUE (SELECT R.Customer_name

FROM Account, Depositor AS R

WHERE T.Customer_name = R.Customer_name AND

R.Account_number = Account.Acount_number ANDAccount.Branch_name = 'Perryridge')

Ta có thể thử sự tồn tại của các bộ trùng trong một vấn tin con bằng cách sử dụng cấu trúc NOT UNIQUE

"Tìm tất cả các khách hàng có ít nhất hai tài khoản ở chi nhánh Perryridge"

SELECT DISTINCT T.Customer_name

FROM Account, Depositor AS T

WHERE NOT UNIQUE (SELECT R.Customer_name

FROM Account, Depositor AS R

WHERE T.Customer_name=R.Customer_name

ANDR.Account_number = Account.Account_number

AND Account.Branch_name = 'Perryridge')

UNIQUE trả lại giá rị false khi và chỉ khi quan hệ có hai bộ trùng nhau. Nếu hai bộ t1, t2 có ít nhất một trường null, phép so sánh t1 = t2 cho kết quả false. Do vậy UNIQUE có thể trả về giá trị true trong khi quan hệ có nhiều bộ trùng nhau nhưng chứa trường giá trị null !

QUAN HỆ DẪN XUẤT

SQL92 cho phép một biểu thức vấn tin con được dùng trong mệnh đề FROM. Nếu biểu thức như vậy được sử dụng, quan hệ kết quả phải

được cho một cái tên và các thuộc tính có thể được đặt tên lại (bằng mệnh đề AS)

Ví dụ câu vấn tin con:

(SELECT Branch_name, AVG(Balance)

FROM Account

GROUP BY Branch_name)

AS result (Branch_name, Avg_balace)

Sinh ra quan hệ gồm tên của tất cả các chi nhánh, và số cân đối trung bình tương ứng. Quan hệ này được đặt tên là result với hai thuộc tính Branch_name và Avg_balance.

"Tìm số cân đối tài sản trung bình của các chi nhánh tại đó số cân đối tài khoản trung bình lớn hơn 1200$"

SELECT Branch_name, avg_balance

FROM (SELECT Branch_name, AVG(Balance)

FROM Account

GROUP BY Branch_name)

AS result (Branch_name, Avg_balace)

WHERE avg_balance > 1200

VIEWS

Trong SQL, để định nghĩa view ta sử dụng lệnh CREATE VIEW. Một view phải có một tên.

CREATE VIEW < tên view > AS < Biểu thức vấn tin >

"Tạo một view gồm các tên chi nhánh, tên của các khách hàng có hoặc một tài khoản hoặc vay ở chi nhánh này"

Giả sử ta muốn đặt tên cho view này là All_customer.

CREATE VIEW All_customerAS

(SELECT Branch_name, Customer_name

FROM Depositor, Account

WHERE Depositor.Account_number = Account.Account_number)

UNION

(SELECT Branch_name, Customer_name

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number)

Tên thuộc tính của một view có thể xác định một cách tường minh như sau:

CREATE VIEW Branch_total_loan (Branch_name, Total_loan) AS

(SELECT Branch_name, sum(Amount)

FROM Loan

GROUP BY Branch_name)

Một view là một quan hệ, nó có thể tham gia vào các câu vấn tin với vai trò của một quan hệ.

SELECT Customer_name

FROM All_customer

WHERE Branch_name = 'Perryridge'

Một câu vấn tin phức tạp sẽ dễ hiểu hơn, dễ viết hơn nếu ta cấu trúc nó bằng cách phân tích nó thành các view nhỏ hơn và sau đó tổ hợp lại.

Định nghĩa view được giữ trong CSDL đến tận khi một lệnh DROP VIEW < tên view > được gọi. Trong chuẩn SQL 3 hiện đang được phát triển bao hàm một đề nghị hỗ trợ những view tạm không được lưu trong CSDL.

SỬA ĐỔI CƠ SỞ DỮ LIỆU

DELETE

INSERT

UPDATE

XÓA (Delete)

Ta chỉ có thể xoá nguyên vẹn một bộ trong một quan hệ, không thể xoá các giá trị của các thuộc tính. Biểu thức xoá trong SQL là:

DELETE FROM r

[WHERE P]

Trong đó p là một vị từ và r là một quan hệ.

Lệnh DELETE duyệt qua tất cả các bộ t trong quan hệ r, nếu P(t) là true, DELETE xoá t khỏi r. Nếu không có mệnh đề WHERE, tất cả các bộ trong r bị xoá.

Lệnh DELETE chỉ hoạt động trên một quan hệ.

- DELETE FROM Loan = Xoá tất cả các bộ của quan hệ Loan
- DELETE FROM Depositor WHERE Customer_name = 'Smith'
- DELETE FROM Loan

WHERE Amount BETWEEN 1300 AND 1500

- DELETE FROM Account

WHERE Branch_name IN (SELECT Branch_name

FROM Branch

WHERE Branch_city = 'Brooklyn')

- DELETE FROM Account

WHERE Balance < (SELECT AVG(Balance)

FROM Account)

XEN (Insert)

Để xen dữ liệu vào một quan hệ, ta xác định một bộ cần xen hoặc viết một câu vấn tin kết quả của nó là một tập các bộ cần xen. Các giá trị thuộc tính của bộ cần xen phải thuộc vào miền giá trị của thuộc tính và số thành phần của bộ phải bằng với ngôi của quan hệ.

“Xen vào quan hệ Account một bộ có số tài khoản là A-9732, số cân đối là 1200$ và tài khoản này được mở ở chi nhánh Perryridge”

INSERT INTO Account

VALUES (‘Perryridge’, ‘A-9732’, 1200);

Trong ví dụ này thự tự các giá trị thuộc tính cần xen trùng khớp với thứ tự các thuộc tính trong sơ đồ quan hệ. SQL cho phép chỉ rõ các thuộc tính và các giá trị tương ứng cần xen:

INSERT INTO Account (Branch_name, Account_number, Balance)

VALUES (‘Perryridge’, ‘A-9732’, 1200);

INSERT INTO Account (Account_number, Balance, Branch_name)

VALUES (‘A-9732’, 1200, ‘Perryridge’);

“Cấp cho tất cả các khách hàng vay ở chi nhánh Perryridge một tài khoản với số cân đối là 200$ như một quà tặng sử dụng số vay như số tài khoản“

INSERT INTO Account

SELECT Branch_name, Loan_number, 200

FROM Loan

WHERE Branch_name = ‘Perryridge’

INSERT INTO Depositor

SELECT Customer_name, Loan_number

FROM Borrower, Loan

WHERE Borrower.Loan_number = Loan.Loan_number AND

Branch_name = ‘Perryridge’

CẬP NHẬT (Update)

Câu lệnh UPDATE cho phép thay đổi giá trị thuộc tính của các bộ

"Thêm lãi hàng năm vào số cân đối với tỷ lệ lãi suất 5%"

UPDATE Account

SET Balance = Balance*1.05

Giả sử các tài khoản có số cân đối > 10000$ được hưởng lãi suất 6%, các tài khoản có số cân đối nhỏ hơn hoặc bằng 10000 được hưởng lãi suất 5%

UPDATE Account

SET Balance = Balance*1.06

WHERE Balance > 10000

UPDATE Account

SET Balance = Balance*1.05

WHERE Balance <= 10000

SQL92 đưa vào cấu trúc CASE như sau:

CASE

WHEN P1 THEN Result1

WHEN P2 THEN Result2

...

WHEN Pn THEN Resultn

ELSEResult0

END

trong đó Pi là các vị từ, Resulti là các kết quả trả về của hoạt động CASE tương ứng với vị từ Pi đầu tiên thỏa mãn. Nếu không vị từ Pi nào thỏa mãn CASE trả về Result0.

Với cấu trúc CASE như vậy ta có thể viết lại yêu cầu trên như sau:

UPDATE Account

SET Balance = CASE

WHEN Balance > 10000 THEN Balance*1.06

ELSE Balance*1.05

END

“Trả 5% lãi cho các tài khoản có số cân đối lớn hơn số cân đối trung bình”

UPDATE Account

SET Balance = Balance*1.05

WHERE Balance > SELECT AVG(Balance)

FROM Account

CÁC QUAN HỆ NỐI

SQL92 cung cấp nhiều cơ chế cho nối các quan hệ bao hàm nối có điều kiện và nối tự nhiên cũng như các dạng của nối ngoài.

Loan INNER JOIN Borrower

ON Loan.Loan_number = Borrower.Loan_number

Nối quan hệ Loan và quan hệ Borrower với điều kiên:

Loan.Loan_number = Borrower.Loan_number

Quan hệ kết quả có các thuộc tính của quan hệ Loan và các thuộc tính của quan hệ Borrower (như vậy thuộc tính Loan_number xuất hiện 2 lần trong quan hệ kết quả).

Để đổi tên quan hệ (kết quả) và các thuộc tính, ta sử dụng mệnh đề AS

Loan INNER JOIN Borrower

ON Loan.Loan_number = Borrower.Loan_number

AS LB(Branch, Loan_number, Amount, Cust, Cust_Loan_number)

Loan LEFT OUTER JOIN Borrower

ON Loan.Loan_number = Borrower.Loan_number

Phép nối ngoài trái được tính như sau: Đầu tiên tính kết quả của nối trong INNER JOIN. Sau đó đối với mỗi bộ t của quan hệ trái (Loan) không tương xứng với bộ nào trong quan hệ bên phải (borrower) khi đó thêm vào kết quả bộ r gồm các giá trị thuộc tính trái là các giá trị thuộc tính của t, các thuộc tính còn lại (phải) được đặt là null.

Loan NATURAL INNER JOIN Borrower

Là nối tự nhiên của quan hệ Loan và quan hệ Borrower (thuộc tính trùng tên là Loan_number).

NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU (DDL)

DDL SQL cho phép đặc tả:

- Sơ đồ cho mỗi quan hệ
- Miền giá trị kết hợp với mỗi thuộc tính
- các ràng buộc toàn vẹn
- tập các chỉ mục được duy trì cho mỗi quan hệ

- thông tin về an toàn và quyền cho mỗi quan hệ
- cấu trúc lưu trữ vật lý của mỗi quan hệ trên đĩa

CÁC KIỂU MIỀN TRONG SQL

SQL-92 hỗ trợ nhiều kiểu miền trong đó bao hàm các kiểu sau:

- char(n) / charater: chuỗi ký tự dộ dài cố định, với độ dài n được xác định bởi người dùng
- vachar(n) / character varying (n): chuỗi ký tự độ dài thay đổi, với độ dài tối đa được xác dịnh bởi người dung là n
- int / integer: tập hữu hạn các số nguyên
- smallint: tập con của tập các số nguyên int
- numeric(p, d): số thực dấu chấm tĩnh gồm p chữ số (kể cả dấu) và d trong p chữ số là các chữ số phần thập phân
- real, double precision: số thực dấu chấm động và số thực dấu chấm động chính xác kép
- float(n): số thực dấu chấm động với độ chính xác được xác định bởi người dùng ít nhất là n chữ số thập phân
- date: kiểu năm tháng ngày (YYYY, MM, DD)
- time: kiểu thời gian (HH, MM, SS)

SQL-92 cho phép định nghĩa miền với cú pháp:

CREATE DOMAIN < tên miền > < Type >

Ví dụ: CREATE DOMAIN hoten char(30);

Sau khi đã định nghĩa miền với tên hoten ta có thể sử dụng nó để định nghĩa kiểu của các thuộc tính

ĐỊNH NGHĨA SƠ ĐỒ TRONG SQL.

Lệnh CREATE TABLE với cú pháp

CREATE TABLE < tên bảng > (

< Thuộc tính 1 >< miền giá trị thuộc tính 1 > ,

...

< Thuộc tính n >< miền giá trị thuộc tính n> ,

< ràng buộc toàn vẹn 1 > ,

...

< ràng buộc toàn vẹn k >)

Các ràng buộc toàn vẹn cho phép bao gồm:

primary key ($A_{i_1}, A_{i_2}, \ldots, A_{i_m}$)

và

check(P)

Đặc tả primary key chỉ ra rằng các thuộc tính $A_{i_1}, A_{i_2}, \ldots, A_{i_m}$ tạo nên khoá chính của quan hệ. Mệnh đề check xác định một vị từ P mà mỗi bộ trong quan hệ phải thoả mãn.

Ví dụ:

CREATE TABLE customer (

customer_nameCHAR(20) not null,

customer_streetCHAR(30),

customer_cityCHAR(30),

PRIMARY KEY(customer_name));

CREATE TABLE branch (

branch_nameCHAR(15) not null,

branch_cityCHAR(30),

assetsINTEGER,

PRIMARY KEY (branch_name),

CHECK (assets >= 0));

CREATE TABLE account (

account_numberCHAR(10) not null,

branch_nameCHAR(15),

balanceINTEGER,

PRIMARY KEY (account_number),

CHECK(balance >= 0));

CREATE TABLE depositor (

customer_nameCHAR(20) not null,

account_numberCHAR(10) not null,

PRIMARY KEY (customer_name, account_namber));

Giá trị null là giá trị hợp lệ cho mọi kiểu trong SQL. Các thuộc tính được khai báo là primary key đòi hỏi phải là not null và duy nhất. do vậy các khai báo not null trong ví dụ trên là dư (trong SQL-92).

CREATE TABLE student (

nameCHAR(15) not null,

student_IDCHAR(10) not null,

degree_levelCHAR(15) not null,

PRIMARY KEY (student_ID),

CHECK (degree_level IN (‘Bachelors’, ‘Masters’, ‘Doctorats’));

- Xoá một quan hệ khỏi CSDL sử dụng lệnh Drop table với cú pháp:

DROP TABLE < tên bảng >

- Thêm thuộc tính vào bảng đang tồn tại sử dụng lệnh Alter table với cú pháp:

ALTER TABLE < tên bảng > ADD < thuộc tính > < miền giá trị >

- Xoá bỏ một thuộc tính khỏi bảng đang tồn tại sử dụng lệnh Alter table với cú pháp:

ALTER TABLE < Tên bảng > DROP < tên thuộc tính >

SQL NHÚNG (Embedded SQL)

Một ngôn ngữ trong đó các vấn tin SQL được nhúng gọi là ngôn ngữ chủ (host language), cấu trúc SQL cho phép trong ngôn ngữ chủ tạo nên SQL nhúng. Chương trình được viết trong ngôn ngữ chủ có thể sử dụng cú pháp SQL nhúng để truy xuất và cập nhật dữ liệu được lưu trữ trong CSDL.

BÀI TẬP CHƯƠNG II

- Xét CSDL bảo hiểm sau:

person(ss#, name, address): Số bảo hiểm ss# sở hữu bởi người tên name ở địa chỉ address

car(license, year, model): Xe hơi số dăng ký license, sản xuất năm year, nhãn hiệu Model

accident(date, driver, damage_amount): tai nạn xảy ra ngày date, do người lái driver, mức hư hại damage_amount

owns(ss#, license): người mang số bảo hiểm ss# sở hữu chiếc xe mang số đăng ký license

log(license, date, driver): ghi sổ chiếc xe mang số đăng ký license, bị tai nạn ngày do người lái driver

các thuộc tính được gạch dưới là các primary key. Viết trong SQL các câu vấn tin sau:

- Tìm tổng số người xe của họ gặp tai nạn năm 2001
- Tìm số các tai nạn trong đó xe của"John"liên quan tới
- Thêm khách hàng mới: ss# =”A-12345”, name ="David”, address ="35 Chevre Road”, license ="109283”, year =”2002”, model ="FORD LASER"vào CSDL
- xoá các thông tin lien quan dến xe model "MAZDA"của"John Smith”
- Thêm thông tin tai nạn cho chiếc xe"TOYOTA"của khách hàng mang số bảo hiểm số"A-84626”
- Xét CSDL nhân viên:

employee (E_name, street, city): Nhân viên có tên E_name, cư trú tại phố street, trong thành phố city

works (E_name, C_name, salary): Nhân viên tên E_name làm việc cho công ty C_name với mức lương salary

copany (C_name, city): Công ty tên C_name đóng tại thành phố city

manages(E_name, M_name): Nhân viên E_name dưới sự quản lý của nhân viên M_name

Viết trong SQL các câu vấn tin sau:

1. Tìm tên của tất cả các nhân viên làm việc cho First Bank

2. Tìm tên và thành phố cư trú của các nhân viên làm việc cho First Bank
3. Tìm tên, phố, thành phố cư trú làm việc cho First Bank hưởng mức lương > 10000$
4. Tìm tất cả các nhân viên trong CSDL sống trong cùng thành phố với công ty mang họ làm việc cho
5. Tìm tất cả các nhân viên sông trong cùng thành phố, cùng phố với người quản lý của họ
6. Tìm trong CSDL các nhân viên không làm việc cho First Bank
7. Tìm trong CSDL, các nhân viên hưởng mức lương cao hơn mọi nhân viên của Small Bank
8. Giả sử một công ty có thể đóng trong một vaì thành phố. Tìm tất cả các công ty đóng trong mỗi thành phố trong đó Small Bank đóng.
9. Tìm tất cả các nhân viên hưởng múc lương cao hơn mức lương trung bình của công ty họ làm việc
10. Tìm công ty có nhiều nhân viên nhất
11. Tìm công ty có tổng số tiền trả lương nhỏ nhất
12. Tìm tất cả các công ty có mức lương trung bình cao hơn mức luong trung bình của công ty First Bank
13. Thay đổi thành phố cư trú của nhân viên"Jones"thành NewTown
14. Nâng lương cho tất cả các nhân viên của First Bank lên 10%
15. nâng lương cho các nhà quản lý của công ty First Bank lên 10%
16. Xoá tất cả các thông tin liên quan tới cong ty Bad Bank

Lưu trữ và cấu trúc tập tin
MỤC ĐÍCH Chương này trình bày các vấn đề liên quan đến vấn đề lưu trữ dữ liệu (trên lưu trữ ngoài, chủ yếu trên đĩa cứng). Việc lưu trữ dữ liệu phải được tổ chức sao cho có thể cất giữ một lượng lớn, có thể rất lớn dữ liệu nhưng quan trọng hơn cả là sự lưu trữ phải cho phép lấy lại dữ liệu cần thiết mau chóng. Các cấu trúc trợ giúp cho truy xuất nhanh dữ liệu được trình bày là: chỉ mục (indice), B+ cây (B+-tree), băm (hashing) ... Các thiết bị lưu trữ (đĩa) có thể bị hỏng hóc không lường trước, các kỹ thuật RAID cho ra một giải pháp hiệu quả cho vấn đề này. YÊU CẦU Hiểu rõ các đặc điểm của các thiết bị lưu trữ, cách tổ chức lưu trữ, truy xuất đĩa. Hiểu rõ nguyên lý và kỹ thuật của tổ chức hệ thống đĩa RAID Hiểu rõ các kỹ thuật tổ chức các mẩu tin trong file Hiểu rõ các kỹ thuật tổ chức file Hiểu và vận dụng các kỹ thuật hỗ trợ tìm lại nhanh thông tin: chỉ mục (được sắp, B+-cây, băm)

KHÁI QUÁT VỀ PHƯƠNG TIỆN LƯU TRỮ VẬT LÝ

Có một số kiểu lưu trữ dữ liệu trong các hệ thống máy tính. Các phương tiện lưu trữ được phân lớp theo tốc độ truy xuất, theo giá cả và theo độ tin cậy của phương tiện. Các phương tiện hiện có là:

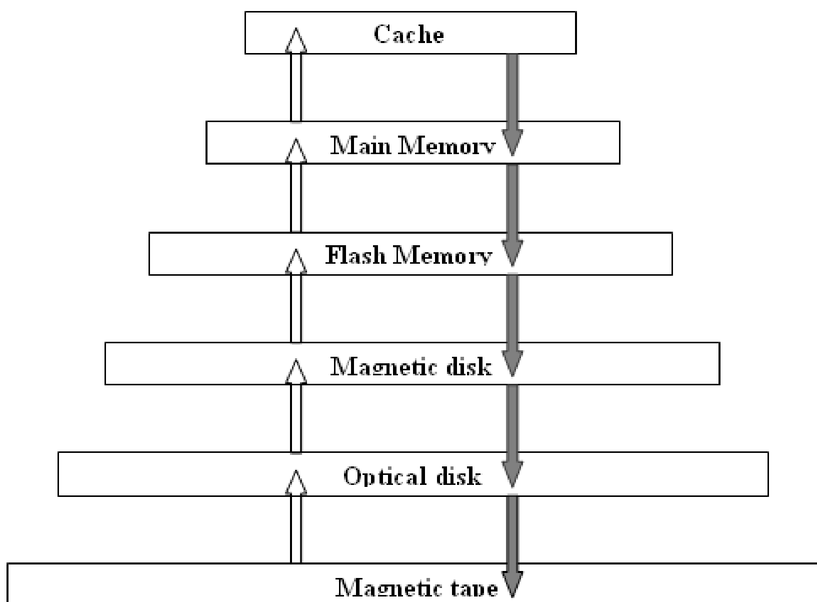
- Cache: là dạng lưu trữ nhanh nhất và cũng đắt nhất trong các phương tiện lưu trữ. Bộ nhớ cache nhỏ; sự sử dụng nó được quản trị bởi hệ điều hành
- Bộ nhớ chính (main memory): Phương tiện lưu trữ dùng để lưu trữ dữ liệu sẵn sàng được thực hiện. Các chỉ thị máy mục đích chung (general-purpose) hoạt động trên bộ nhớ chính. Mặc dầu bộ nhớ chính có thể chứa nhiều megabytes dữ liệu, nó vẫn là quá nhỏ (và quá đắt giá) để lưu trữ toàn bộ một cơ sở dữ liệu. Nội dung trong bộ nhớ chính thường bị mất khi mất cấp nguồn
- Bộ nhớ Flash: Được biết như bộ nhớ chỉ đọc có thể lập trình, có thể xoá (EEPROM: Electrically Erasable Programmable Read-Only Memory), Bộ nhớ Flash khác bộ nhớ chính ở chỗ dữ liệu còn tồn tại trong bộ nhớ flash khi mất cấp nguồn. Đọc dữ liệu từ bộ nhớ flash mất ít hơn 100 ns , nhanh như đọc dữ liệu từ bộ nhớ chính. Tuy nhiên, viết dữ liệu vào bộ nhớ flash phức tạp hơn nhiều. Dữ liệu được viết (một lần mất khoảng 4 đến 10 s) nhưng không thể viết đè trực tiếp. Để viết đè bộ nhớ đã được viết, ta phải xoá trắng toàn bộ bộ nhớ sau đó mới có thể viết lên nó.
- Lưu trữ đĩa từ (magnetic-disk): (ở đây, được hiểu là đĩa cứng) Phương tiện căn bản để lưu trữ dữ liệu trực tuyến, lâu dài. Thường toàn bộ cơ sở dữ liệu được lưu trữ trên đĩa từ. Dữ liệu phải được chuyển từ đĩa vào bộ nhớ chính trước khi được truy nhập. Khi dữ liệu trong bộ nhớ chính này bị sửa đổi, nó phải được viết lên đĩa. Lưu trữ đĩa được xem là truy xuất trực tiềp vì có thể đọc dữ liệu trên đĩa theo một thứ tự bất kỳ. Lưu trữ đĩa vẫn tồn tại khi mất cấp nguồn. Lưu trữ đĩa có thể bị hỏng hóc, tuy không thường xuyên.
- Lưu trữ quang (Optical storage): Dạng quen thuộc nhất của đĩa quang học là loại đĩa CD-ROM : Compact-Disk Read-Only Memory. Dữ liệu được lưu trữ trên các đĩa quang học được đọc bởi laser. Các đĩa quang học CD-ROM chỉ có thể dọc. Các phiên bản khác của chúng là loại đĩa quang học: viết một lần, đọc nhiều lần (write-once, read-

many: WORM) cho phép viết dữ liệu lên đĩa một lần, không cho phép xoá và viết lại, và các đĩa có thể viết lại (rewritable) v..v
- Lưu trữ băng từ (tape storage): Lưu trữ băng từ thường dùng để backup dữ liệu. Băng từ rẻ hơn đĩa, truy xuất dữ liệu chậm hơn (vì phải truy xuất tuần tự). Băng từ thường có dung lượng rất lớn.

Các phương tiện lưu trữ có thể được tổ chức phân cấp theo tốc độ truy xuất và giá cả. Mức cao nhất là nhanh nhất nhưng cũng là đắt nhất, giảm dần xuống các mức thấp hơn.

Các phương tiện lưu trữ nhanh (cache, bộ nhớ chính) được xem như là lưu trữ sơ cấp (primary storage), các thiết bị lưu trữ ở mức thấp hơn như đĩa từ được xem như lưu trữ thứ cấp hay lưu trữ trực tuyến (on-line storage), còn các thiết bị lưu trữ ở mức thấp nhất và gần thấp nhất như đĩa quang học, băng từ kể cả các đĩa mềm được xếp vào lưu trữ tam cấp hay lưu trữ không trực tuyến (off-line).

Bên cạnh vấn đề tốc độ và giá cả, ta còn phải xét đến tính lâu bền của các phương tiện lưu trữ.



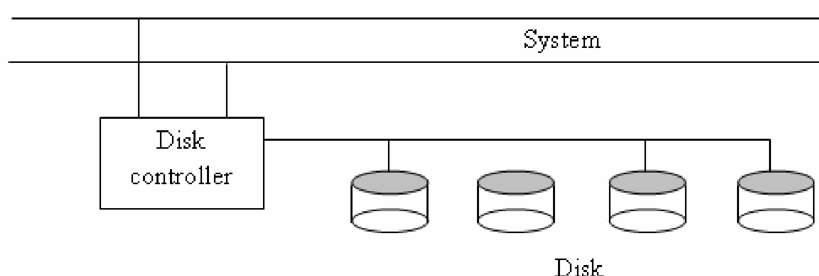
Phân cấp thiết bị lưu trữ

ĐĨA TỪ

ĐẶC TRƯNG VẬT LÝ CỦA ĐĨA

Mỗi tấm đĩa có dạng hình tròn, hai mặt của nó được phủ bởi vật liệu từ tính, thông tin được ghi trên bề mặt đĩa. Đĩa gồm nhiều tấm đĩa. Ta sẽ sử dụng thuật ngữ đĩa để chỉ các đĩa cứng.

Khi đĩa được sử dụng, một động cơ ổ đĩa làm quay nó ở một tốc độ không đổi. Một đầu đọc-viết được định vị trên bề mặt của tấm đĩa. Bề mặt tấm đĩa được chia logic thành các rãnh, mỗi rãnh lại được chia thành các sector, một sector là một đơn vị thông tin nhỏ có thể được đọc, viết lên đĩa. Tuỳ thuộc vào kiểu đĩa, sector thay đổi từ 32 bytes đến 4095 bytes, thông thường là 512 bytes. Có từ 4 đến 32 sectors trên một rãnh, từ 20 đén 1500 rãnh trên một bề mặt. Mỗi bề mặt của một tấm đĩa có một đầu đọc viết, nó có thể chạy dọc theo bán kính đĩa để truy cập đến các rãnh khác nhau. Một đĩa gồm nhiều tấm đĩa, các đầu đọc-viết của tất cả các rãnh được gắn vào một bộ được gọi là cánh tay đĩa, di chuyển cùng nhau. Các tấm đĩa được gắn vào một trục quay. Vì các đầu đọc-viết trên các tấm đĩa di chuyển cùng nhau, nên khi đầu đọc-viết trên một tấm đĩa đang ở rãnh thứ i thì các đầu đọc-viết của các tấm đĩa khác cũng ở rãnh thứ i , do vậy các rãnh thứ i của tất cả các tấm đĩa được gọi là trụ (cylinder) thứ i . Một bộ điều khiển đĩa -- giao diện giữa hệ thống máy tính và phần cứng hiện thời của ổ đĩa. Nó chấp nhận các lệnh mức cao để đọc và viết một sector, và khởi động các hành động như di chuyển cánh tay đĩa đến các rãnh đúng và đọc viết dữ liệu. bộ điều khiển đĩa cũng tham gia vào checksum mỗi sector được viết. Checksum được tính từ dữ liệu được viết lên sector. Khi sector được đọc lại, checksum được tính lại từ dữ liệu được lấy ra và so sánh với checksum đã lưu trữ. Nếu dữ liệu bị sai lạc, checksum được tính sẽ không khớp với checksum đã lưu trữ. Nếu lỗi như vậy xảy ra, bộ điều khiển sẽ lặp lại việc đọc vài lần, nếu lỗi vẫn xảy ra, bộ điều khiển sẽ thông báo việc đọc thất bại. Bộ điều khiển đĩa còn có chức năng tái ánh xạ các sector xấu: ánh xạ các sector xấu đến một vị trí vật lý khác. Hình dưới bày tỏ các đĩa được nối với một hệ thống máy tính:



Các đĩa được nối với một hệ thống máy tính hoặc một bộ điều khiển đĩa qua một sự hợp nhất tốc độ cao. Hợp nhất hệ thống máy tính nhỏ (Small Computer-System Interconnect: SCSI) thường được sử dụng để nối kết các đĩa với các máy tính cá nhân và workstation. Mainframe và các hệ thống server thường có các bus nhanh hơn và đắt hơn để nối với các đĩa.

Các đầu đọc-viết được giữ sát với bề mặt đĩa như có thể để tăng độ dày đặc (density).

Đĩa đầu cố định (Fixed-head) có một đầu riêng biệt cho mỗi rãnh, sự sắp xếp này cho phép máy tính chuyển từ rãnh này sang rãnh khác mau chóng, không phải di chuyển đầu đọc-viết. Tuy nhiên, cần một số rất lớn đầu đọc-viết, điều này làm nâng giá của thiết bị.

ĐO LƯỜNG HIỆU NĂNG CỦA ĐĨA

Các tiêu chuẩn đo lường chất lượng chính của đĩa là dung lượng, thời gian truy xuất, tốc độ truyền dữ liệu và độ tin cậy.

- Thời gian truy xuất (access time): là khoảng thời gian từ khi yêu cầu đọc/viết được phát đi đến khi bắt đầu truyền dữ liệu. Để truy xuất dữ liệu trên một sector đã cho của một đĩa, đầu tiên cánh tay đĩa phải di chuyển đến rãnh đúng, sau đó phải chờ sector xuất hiện dưới nó, thời gian để định vị cánh tay được gọi là thời gian tìm kiếm (seek time), nó tỷ lệ với khoảng cách mà cánh tay phải di chuyển, thời gian tìm kiếm nằm trong khoảng 2..30 ms tuỳ thuộc vào rãnh xa hay gần vị trí cánh tay hiện tại.
- Thời gian tìm kiếm trung bình (average seek time): Thời gian tìm kiếm trung bình là trung bình của thời gian tìm kiếm, được đo luờng trên một dãy các yêu cầu ngẫu nhiên (phân phối đều), và bằng khoảng 1/3 thời gian tìm kiếm trong trường hợp xấu nhất.

- Thời gian tiềm ẩn luân chuyển (rotational latency time): Thời gian chờ sector được truy xuất xuất hiện dưới đầu đọc/viết. Tốc độ quay của đĩa nằm trong khoảng 60..120 vòng quay trên giây, trung bình cần nửa vòng quay để sector cần thiết nằm dưới đầu đọc/viết. Như vậy, thời gian tiềm ẩn trung bình (average latency time) bằng nửa thời gian quay một vòng đĩa.

Thời gian truy xuất bằng tổng của thời gian tìm kiếm và thời gian tiềm ẩn và nằm trong khoảng 10..40 ms.

- Tốc độ truyền dữ liệu: là tốc độ dữ liệu có thể được lấy ra từ đĩa hoặc được lưu trữ vào đĩa. Hiện nay tốc này vào khoảng1..5 Mbps

- Thời gian trung bình không sự cố (mean time to failure): lượng thời gian trung bình hệ thống chạy liên tục không có bất kỳ sự cố nào. Các đĩa hiện nay có thời gian không sự cố trung bình khoảng 30000 .. 800000 giờ nghĩa là khoảng từ 3,4 đến 91 năm.

TỐI ƯU HÓA TRUY XUẤT KHỐI ĐĨA (disk-block)

Yêu cầu I/O đĩa được sinh ra cả bởi hệ thống file lẫn bộ quản trị bộ nhớ ảo trong hầu hết các hệ điều hành. Mỗi yêu cầu xác định địa chỉ trên đĩa được tham khảo, địa chỉ này ở dạng số khối. Một khối là một dãy các sector kề nhau trên một rãnh. Kích cỡ khối trong khoảng 512 bytes đến một vài Kbytes. Dữ liệu được truyền giữa đĩa và bộ nhớ chính theo đơn vị

khối. Mức thấp hơn của bộ quản trị hệ thống file sẽ chuyển đổi địa chỉ khối sang số của trụ, của mặt và của sector ở mức phần cứng.

Truy xuất dữ liệu trên đĩa chậm hơn nhiều so với truy xuất dữ liệu trong bộ nhớ chính, do vậy cần thiết một chiến lược nhằm nâng cao tốc độ truy xuất khối đĩa. Dưới đây ta sẽ thảo luận một vài kỹ thuật nhằm vào mục đích đó.

- Scheduling: Nếu một vài khối của một trụ cần được truyền từ đĩa vào bộ nhớ chính, ta có thể tiết kiệm thời gian truy xuất bởi yêu cầu các khối theo thứ tự mà nó chạy qua dưới đầu đọc/viết. Nếu các khối mong muốn ở trên các trụ khác nhau, ta yêu cầu các khối theo thứ tự sao cho làm tối thiểu sự di chuyển cánh tay đĩa. Các thuật toán scheduling cánh tay đĩa (Disk-arm-scheduling) nhằm lập thứ tự truy xuất các rãnh theo cách làm tăng số truy xuất có thể được xử lý. Một thuật toán thường dùng là thuật toán thang máy (elevator algorithm): Giả sử ban đầu cánh tay di chuyển từ rãnh trong nhất hướng ra phía ngoài đĩa, đối với mỗi rãnh có yêu cầu truy xuất, nó dừng lại, phục vụ yêu cầu đối với rãnh này, sau đó tiếp tục di chuyển ra phía ngoài đến tận khi không có yêu cầu nào chờ các rãnh xa hơn phía ngoài. Tại điểm này, cánh tay đổi hướng, di chuyển vào phía trong, lại dừng lại trên các rãnh được yêu cầu, và cứ như vậy đến tận khi không còn rãnh nào ở trong hơn được yêu cầu, rồi lại đổi hướng .. v .. v .. Bộ điều khiển đĩa thường làm nhiệm vụ sắp xếp lại các yêu cầu đọc để cải tiến hiệu năng.
- Tổ chức file: Để suy giảm thời gian truy xuất khối, ta có thể tổ chức các khối trên đĩa theo cách tương ứng gần nhất với cách mà dữ liệu được truy xuất. Ví dụ, Nếu ta muốn một file được truy xuất tuần tự, khi đó ta bố trí các khối của file một cách tuần tự trên các trụ kề nhau. Tuy nhiên việc phân bố các khối lưu trữ kề nhau này sẽ bị phá vỡ trong quá trình phát triển của file   file không thể được phân bố trên các khối kề nhau được nữa, hiện tượng này dược gọi là sự phân mảnh (fragmentation). Nhiều hệ điều hành cung cấp tiện ích giúp suy giảm sự phân mảnh này (Defragmentation) nhằm làm tăng hiệu năng truy xuất file.
- Các buffers viết không hay thay đổi: Vì nội dung của bộ nhớ chính bị mất khi mất nguồn, các thông tin về cơ sở dữ liệu cập nhật phải được ghi lên đĩa nhằm đề phòng sự cố. Hiệu năng của các ứng dụng cập nhật cường độ cao phụ thuộc mạnh vào tốc độ viết đĩa. Ta có thể sử dụng bộ nhớ truy xuất ngẫu nhiên không hay thay đổi (nonvolatile RAM) để nâng tốc độ viết đĩa. Nội dung của nonvolatile RAM không bị mất khi mất nguồn. Một phương pháp chung để thực hiện nonvolatile RAM là sử dụng RAM pin dự phòng (battery-back-up RAM). Khi cơ sở dữ liệu yêu cầu viết một khối lên đĩa, bộ điều khiển dĩa viết khối này lên buffer nonvolatile RAM, và thông báo ngay cho hệ điều hành là việc viết đã thành công. Bộ điều khiển sẽ viết dữ liệu đến đích của nó trên đĩa, mỗi khi đĩa rảnh hoặc buffer nonvolatile RAM đầy. Khi hệ cơ sở dữ liệu yêu cầu một viết khối, nó chỉ chịu một khoảng lặng chờ đợi khi buffer nonvolatile RAM đầy.
- Đĩa log (log disk): Một cách tiếp cận khác để làm suy giảm tiềm năng viết là sử dụng log-disk: Một đĩa được tận hiến cho việc viết một log tuần tự. Tất cả các truy xuất đến log-disk là tuần tự, nhằm loại bỏ thời gian tìm kiếm, và một vài khối kề có thể

được viết một lần, tạo cho viết vào log-disk nhanh hơn viết ngẫu nhiên vài lần. Cũng như trong trường hợp sử dụng nonvolatile RAM, dữ liệu phải được viết vào vị trí hiện thời của chúng trên đĩa, nhưng việc viết này có thể được tiến hành mà hệ cơ sở dữ liệu không cần thiết phải chờ nó hoàn tất. Log-disk có thể được sử dụng để khôi phục dữ liệu. Hệ thống file dựa trên log là một phiên bản của cách tiếp cận log-disk: Dữ liệu không được viết lại lên đích gốc của nó trên đĩa; thay vào đó, hệ thống file lưu vết nơi các khối được viết mới đây nhất trên log-disk, và hoàn lại chúng từ vị trí này. Log-disk được "cô đặc" lại (compacting) theo một định kỳ. Cách tiếp cận này cải tiến hiệu năng viết, song sinh ra sự phân mảnh đối với các file được cập nhật thường xuyên.

RAID

Trong một hệ thống có nhiều đĩa, ta có thể cải tiến tốc độ đọc viết dữ liệu nếu cho chúng hoạt động song song. Mặt khác, hệ thống nhiều đĩa còn giúp tăng độ tin cậy lưu trữ bằng cách lưu trữ dư thừa thông tin trên các đĩa khác nhau, nếu một đĩa có sự cố dữ liệu cũng không bị mất. Một sự đa dạng các kỹ thuật tổ chức đĩa, được gọi là RAID (Redundant Arrays of Inexpensive Disks), được đề nghị nhằm vào vấn đề tăng cường hiệu năng và độ tin cậy.

CẢI TIẾN ĐỘ TIN CẬY THÔNG QUA SỰ DƯ THỪA

Giải pháp cho vấn đề độ tin cậy là đưa vào sự dư thừa: lưu trữ thông tin phụ, bình thường không cần thiết, nhưng nó có thể được sử dụng để tái tạo thông tin bị mất khi gặp sự cố hỏng hóc đĩa, như vậy thời gian trung bình không sự cố tăng lên (xét tổng thể trên hệ thống đĩa).

Đơn giản nhất, là làm bản sao cho mỗi đĩa. Kỹ thuật này được gọi là mirroring hay shadowing. Một đĩa logic khi đó bao gồm hai đĩa vật lý, và mỗi việc viết được thực hiện trên cả hai đĩa. Nếu một đĩa bị hư, dữ liệu có thể được đọc từ đĩa kia. Thời gian trung bình không sự cố của đĩa mirror phụ thuộc vào thời gian trung bình không sự cố của mỗi đĩa và phụ thuộc vào thời gian trung bình được sửa chữa (mean time to repair): thời gian trung bình để một đĩa bị hư được thay thế và phục hồi dữ liệu trên nó.

CẢI TIẾN HIỆU NĂNG THÔNG QUA SONG SONG

Với đĩa mirror, tốc độ đọc có thể tăng lên gấp đôi vì yêu cầu đọc có thể được gửi đến cả hai đĩa. Với nhiều đĩa, ta có thể cải tiến tốc độ truyền bởi phân nhỏ (striping data) dữ liệu qua nhiều đĩa. Dạng đơn giản nhất là tách các bít của một byte qua nhiều đĩa, sự phân nhỏ này được gọi là sự phân nhỏ mức bit (bit-level striping). Ví dụ, ta có một dàn 8 đĩa, ta viết bít thứ i của một byte lên đĩa thứ i . dàn 8 đĩa này có thể được xử lý như một đĩa với các

sector 8 lần lớn hơn kích cỡ thông thường, quan trọng hơn là tốc dộ truy xuất tăng lên tám lần. Trong một tổ chức như vậy, mỗi đĩa tham gia vào mỗi truy xuất (đọc/viết), như vậy, số các truy xuất có thể được xử lý trong một giây là tương tự như trên một đĩa, nhưng mỗi truy xuất có thể đọc/viết nhiều dữ liệu hơn tám lần.

Phân nhỏ mức bit có thể được tổng quát cho số đĩa là bội hoặc ước của 8, Ví dụ, ta có một dàn 4 đĩa, ta sẽ phân phối bít thứ i và bít thứ 4+i vào đĩa thứ i. Hơn nữa, sự phân nhỏ không nhất thiết phải ở mức bit của một byte. Ví dụ, trong sự phân nhỏ mức khối, các khối của một file được phân nhỏ qua nhiều đĩa, với n đĩa, khối thứ i có thể được phân phối qua đĩa (i mod n) + 1. Ta cũng có thể phân nhỏ ở mức byte, sector hoặc các sector của một khối. Hai đích song song trong một hệ thống đĩa là:

1. Nạp nhiều truy xuất nhỏ cân bằng (truy xuất trang) sao cho lượng dữ liệu được nạp trong một đơn vị thời gian của truy xuất như vậy tăng lên.
2. Song song hoá các truy xuất lớn sao cho thời gian trả lời các truy xuất lớn giảm.

CÁC MỨC RAID

Mirroring cung cấp độ tin cậy cao, nhưng đắt giá. Phân nhỏ cung cấp tốc độ truyền dữ liệu cao, nhưng không cải tiến được độ tin cậy. Nhiều sơ đồ cung cấp sự dư thừa với giá thấp bằng cách phối hợp ý tưởng của phân nhỏ với "parity" bit. Các sơ đồ này có sự thoả hiệp giá-hiệu năng khác nhau và được phân lớp thành các mức được gọi là các mức RAID.

Mức RAID 0 : Liên quan đến các dàn đĩa với sự phân nhỏ mức khối, nhưng không có một sự dư thừa nào.

Mức RAID 1 : Liên quan đến mirror đĩa

Mức RAID 2 : Cũng được biết dưới cái tên mã sửa lỗi kiểu bộ nhớ (memory-style error-correcting-code : ECC). Hệ thống bộ nhớ thực hiện phát hiện lỗi bằng bit parity. Mỗi byte trong hệ thống bộ nhớ có thể có một bit parity kết hợp với nó. Sơ đồ sửa lỗi lưu hai hoặc nhiều hơn các bit phụ, và có thể dựng lại dữ liệu nếu một bit bị lỗi. ý tưởng của mã sửa lỗi có thể được sử dụng trực tiếp trong dàn đĩa thông qua phân nhỏ byte qua các đĩa. Ví dụ, bít đầu tiên của mỗi byte có thể được lưu trên đĩa 1, bit thứ hai trên đĩa 2, và cứ như vậy, bit thứ 8 trên đĩa 8, các bit sửa lỗi được lưu trên các đĩa thêm vào. Nếu một trong các đĩa bị hư, các bít còn lại của byte và các bit sửa lỗi kết hợp được đọc từ các đĩa khác có thể giúp tái tạo bít bị mất trên đĩa hư, như vậy ta có thể dựng lại dữ liệu. Với một dàn 4 đĩa dữ liệu, RAID mức 2 chỉ cần thêm 3 đĩa để lưu các bit sửa lỗi (các đĩa thêm vào này được gọi là các đĩa overhead), so sánh với RAID mức 1, cần 4 đĩa overhead.

Mức RAID 3 : Còn được gọi là tổ chức parity chen bit (bit-interleaved parity). Bộ điều khiển đĩa có thể phát hiện một sector được đọc đúng hay sai, như vậy có thể sử dụng chỉ một bit parity để sửa lỗi: Nếu một trong các sector bị hư, ta biết chính xác đó là sector nào, Với mỗi bit trong sector này ta có thể hình dung nó là bít 1 hay bit 0 bằng cách tính parity

của các bit tương ứng từ các sector trên các đĩa khác. Nếu parity của các bit còn lại bằng với parity được lưu, bit mất sẽ là 0, ngoài ra bit mất là 1. RAID mức 3 tốt như mức 2 nhưng it tốn kém hơn (chỉ cần một đĩa overhead).

Mức RAID 4 : Còn được gọi là tổ chức parity chen khối (Block-interleaved parity), lưu trữ các khối đúng như trong các đĩa chính quy, không phân nhỏ chúng qua các đĩa nhưng lấy một khối parity trên một đĩa riêng biệt đối với các khối tương ứng từ N đĩa khác. Nếu một trong các đĩa bị hư, khối parity có thể được dùng với các khối tương ứng từ các đĩa khác để khôi phục khối của đĩa bị hư.

Một đọc khối chỉ truy xuất một đĩa, cho phép các yêu cầu khác được xử lý bởi các đĩa khác. Như vậy, tốc độ truyền dữ liệu đối với mỗi truy xuất chậm, nhưng nhiều truy xuất đọc có thể được xử lý song song, dẫn đến một tốc độ I/O tổng thể cao hơn. Tốc độ truyền đối với các đọc dữ liệu lớn (nhiều khối) cao do tất cả các đĩa có thể được đọc song song; các viết dữ liệu lớn (nhiều khối) cũng có tốc độ truyền cao vì dữ liệu và parity có thể được viết song song. Tuy nhiên, viết một khối đơn phải truy xuất đĩa trên đó khối được lưu trữ, và đĩa parity (do khối parity cũng phải được cập nhật). Như vậy, viết một khối đơn yêu cầu 4 truy xuất: hai để đọc hai khối cũ, và hai để viết lại hai khối.

Mức RAID 5 : Còn gọi là parity phân bố chen khối (Block-interleaved Distributed Parity), cải tiến của mức 4 bởi phân hoạch dữ liệu và parity giữa toàn bộ N+1 đĩa, thay vì lưu trữ dữ liệu trên N đĩa và parity trên một đĩa riêng biệt như trong RAID 4. Trong RAID 5, tất cả các đĩa có thể tham gia làm thoả mãn các yêu cầu đọc, như vậy sẽ làm tăng tổng số yêu cầu có thể được đặt ra trong một đơn vị thời gian. Đối với mỗi khối, một đĩa lưu trữ parity, các đĩa khác lưu trữ dữ liệu. Ví dụ, với một dàn năm đĩa, parity đối với khối thứ n được lưu trên đĩa (n mod 5)+1. Các khối thứ n của 4 đĩa khác lưu trữ dữ liệu hiện hành của khối đó.

Mức RAID 6 : Còn được gọi là sơ đồ dư thừa P+Q (P+Q redundancy scheme), nó rất giống RAID 5 nhưng lưu trữ thông tin dư thừa phụ để canh chừng nhiều đĩa bị hư. Thay vì sử dụng parity, người ta sử dụng các mã sửa lỗi.

CHỌN MỨC RAID ĐÚNG

Nếu đĩa bị hư, Thời gian tái tạo dữ liệu của nó là đáng kể và thay đổi theo mức RAID được dùng. Sự tái tạo dễ dàng nhất đối với mức RAID 1. Đối với các mức khác, ta phải truy xuất tất cả các đĩa khác trong dàn đĩa để tái tạo dữ liệu trên đĩa bị hư. Hiệu năng tái tạo của một một hệ thống RAID có thể là một nhân tố quan trọng nếu việc cung cấp dữ liệu liên tục được yêu cầu (thường xảy ra trong các hệ CSDL hiệu năng cao hoặc trao đổi). Hơn nữa, hiệu năng tái tạo ảnh hưởng đến thời gian trung bình không sự cố.

Vì RAID mức 2 và 4 được gộp lại bởi RAID mức 3 và 5, Việc lựa chọn mức RAID thu hẹp lại trên các mức RAID còn lại. Mức RAID 0 được dùng trong các ứng dụng hiệu năng

cao ở đó việc mất dữ liệu không có gì là trầm trọng cả. RAID mức 1 là thông dụng cho các ứng dụng lưu trữ các log-file trong hệ CSDL. Do mức 1 có overhead cao, mức 3 và 5 thường được ưa thích hơn đối với việc lưu trữ khối lượng dữ liệu lớn. Sự khác nhau giữa mức 3 và mức 5 là tốc độ truyền dữ liệu đối lại với tốc độ I/O tổng thể. Mức 3 được ưa thích hơn nếu truyền dữ liệu cao được yêu cầu, mức 5 được ưa thích hơn nếu việc đọc ngẫu nhiên là quan trọng. Mức 6, tuy hiện nay ít được áp dụng, nhưng nó có độ tin cậy cao hơn mức 5.

MỞ RỘNG

Các quan niệm của RAID được khái quát hoá cho các thiết bị lưu trữ khác, bao hàm các dàn băng, thậm chí đối với quảng bá dữ liệu trên các hệ thống không dây. Khi áp dụng RAID cho dàn băng, cấu trúc RAID cho khả năng khôi phục dữ liệu cả khi một trong các băng bị hư hại. Khi áp dụng đối với quảng bá dữ liệu, một khối dữ liệu được phân thành các đơn vị nhỏ và được quảng bá cùng với một đơn vị parity; nếu một trong các đơn vị này không nhận được, nó có thể được dựng lại từ các đơn vị còn lại.

LƯU TRỮ TAM CẤP (tertiary storage)

ĐĨA QUANG HỌC

CR-ROM có ưu điểm là có khả năng lưu trữ lớn, dễ di chuyển (có thể đưa vào và lấy ra khỏi ổ đĩa như đĩa mềm), hơn nữa giá lại rẻ. Tuy nhiên, so với ổ đĩa cứng, thời gian tìm kiếm của ổ CD-ROM chậm hơn nhiều (khoảng 250ms), tốc độ quay chậm hơn (khoảng 400rpm), từ đó dẫn đến độ trễ cao hơn; tốc độ truyền dữ liệu cũng chậm hơn (khoảng 150Kbytes/s). Gần đây, một định dạng mới của đĩa quang học - Digital video disk (DVD) - được chuẩn hoá, các đĩa này có dung lượng trong khoảng 4,7GBytes đến 17 GBytes. Các đĩa WORM, REWRITABLE cũng trở thành phổ biến. Các WORM jukeboxes là các thiết bị có thể lưu trữ một số lớn các đĩa WORM và có thể nạp tự động các đĩa theo yêu cầu đến một hoặc một vài ổ WORM.

BĂNG TỪ

Băng từ có thể lưu một lượng lớn dữ liệu, tuy nhiên, chậm hơn so với đĩa từ và đĩa quang học. Truy xuất băng buộc phải là truy xuất tuần tự, như vậy nó không thích hợp cho hầu hết các đòi hỏi của lưu trữ thứ cấp. Băng từ được sử dụng chính cho việc backup, cho lưu trữ các thông tin không được sử dụng thường xuyên và như một phương tiện ngoại vi (off-line medium) để truyền thông tin từ một hệ thống đến một hệ thống khác. Thời gian để định vị đoạn băng lưu dữ liệu cần thiết có thể kéo dài đến hàng phút. Jukeboxes băng chứa một lượng lớn băng, với một vài ổ băng và có thể lưu trữ được nhiều TeraBytes (1012 Bytes)

TRUY XUẤT LƯU TRỮ

Một cơ sở dữ liệu được ánh xạ vào một số các file khác nhau được duy trì bởi hệ điều hành nền. Các file này lưu trú thường trực trên các đĩa với backup trên băng. Mỗi file được phân hoạch thành các đơn vị lưu trữ độ dài cố định được gọi là khối - đơn vị cho cả cấp phát lưu trữ và truyền dữ liệu.

Một khối có thể chứa một vài hạng mục dữ liệu (data item). Ta giả thiết không một hạng mục dữ liệu nào trải ra trên hai khối. Mục tiêu nổi trội của hệ CSDL là tối thiểu hoá số khối truyền giữa đĩa và bộ nhớ. Một cách để giảm số truy xuất đĩa là giữ nhiều khối như có thể trong bộ nhớ chính. Mục đích là để khi một khối được truy xuất, nó đã nằm sẵn trong bộ nhớ chính và như vậy không cần một truy xuất đĩa nào cả.

Do không thể lưu tất cả các khối trong bộ nhớ chính, ta cần quản trị cấp phát không gian sẵn có trong bộ nhớ chính để lưu trữ các khối. Bộ đệm (Buffer) là một phần của bộ nhớ chính sẵn có để lưu trữ bản sao khối đĩa. Luôn có một bản sao trên đĩa cho mỗi khối, song các bản sao trên đĩa của các khối là các phiên bản cũ hơn so với phiên bản trong buffer. Hệ thống con đảm trách cấp phát không gian buffer được gọi là bộ quản trị buffer.

BỘ QUẢN TRỊ BUFFER

Các chương trình trong một hệ CSDL đưa ra các yêu cầu cho bộ quản trị buffer khi chúng cần một khối đĩa. Nếu khối này đã sẵn sàng trong buffer, địa chỉ khối trong bộ nhớ chính được chuyển cho người yêu cầu. Nếu khối chưa có trong buffer, bộ quản trị buffer đầu tiên cấp phát không gian trong buffer cho khối, rút ra một số khối khác, nếu cần thiết, để lấy không gian cho khối mới. Khối được rút ra chỉ được viết lại trên đĩa khi nó có bị sửa đổi kể từ lần được viết lên đĩa gần nhất. Sau đó bộ quản trị buffer đọc khối từ đĩa vào buffer, và chuyển địa chỉ của khối trong bộ nhớ chính cho người yêu cầu. Bộ quản trị buffer không khác gì nhiều so với bộ quản trị bộ nhớ ảo, một điểm khác biệt là kích cỡ của một CSDL có thể rất lớn không đủ chứa toàn bộ trong bộ nhớ chính do vậy bộ quản trị buffer phải sử dụng các kỹ thuật tinh vi hơn các sơ đồ quản trị bộ nhớ ảo kiểu mẫu.

Chiến lược thay thế. Khi không có chỗ trong buffer, một khối phải được xoá khỏi buffer trước khi một khối mới được đọc vào. Thông thường, hệ điều hành sử dụng sơ đồ LRU (Least Recently Used) để viết lên đĩa khối ít được dùng gần đây nhất, xoá bỏ nó khỏi buffer. Cách tiếp cận này có thể được cải tiến đối với ứng dụng CSDL.

Khối chốt (pinned blocks). Để hệ CSDL có thể khôi phục sau sự cố, cần thiết phải hạn chế thời gian khi viết lại lên đĩa một khối. Một khối không cho phép viết lại lên đĩa được gọi là khối chốt.

Xuất ra bắt buộc các khối (Forced output of blocks). Có những tình huống trong đó cần phải viết lại một khối lên đĩa, cho dù không gian buffer mà nó chiếm là không cần đến. Việc viết này được gọi là sự xuất ra bắt buộc của một khối. Lý do ngắn gọn của yêu cầu

xuất ra bắt buộc khối là nội dung của bộ nhớ chính bị mất khi có sự cố, ngược lại dữ liệu trên dĩa còn tồn tại sau sự cố.

CÁC ĐỐI SÁCH THAY THẾ BUFFER (Buffer-Replacement Policies).

Mục đích của chiến lược thay thế khối trong buffer là tối thiểu hoá các truy xuất đĩa. Các hệ điều hành thường sử dụng chiến lược LRU để thay thế khối. Tuy nhiên, một hệ CSDL có thể dự đoán mẫu tham khảo tương lai. Yêu cầu của một người sử dụng đối với hệ CSDL bao gồm một số bước. Hệ CSDL có thể xác định trước những khối nào sẽ là cần thiết bằng cách xem xét mỗi một trong các bước được yêu cầu để thực hiện hoạt động được yêu cầu bởi người sử dụng. Như vậy, khác với hệ điều hành, hệ CSDL có thể có thông tin liên quan đến tương lai, chí ít là tương lai gần. Trong nhiều trường hợp, chiến lược thay thế khối tối ưu cho hệ CSDL lại là MRU (Most Recently Used): Khối bị thay thế sẽ là khối mới được dùng gần đây nhất!

Bộ quản trị buffer có thể sử dụng thông tin thống kê liên quan đến xác suất mà một yêu cầu sẽ tham khảo một quan hệ riêng biệt nào đó. Tự điển dữ liệu là một trong những phần được truy xuất thường xuyên nhất của CSDL. Như vậy, bộ quản trị buffer sẽ không nên xoá các khối tự điển dữ liệu khỏi bộ nhớ chính trừ phi các nhân tố khác bức chế làm điều đó. Một chỉ mục (Index) đối với một file được truy xuất thường xuyên hơn chính bản thân file, vậy thì bộ quản trị buffer cũng không nên xoá khối chỉ mục khỏi bộ nhớ chính nếu có sự lựa chọn.

Chiến luợc thay thế khối CSDL lý tưởng cần hiểu biết về các hoạt động CSDL đang được thực hiện. Không một chiến lược đơn lẻ nào được biết nắm bắt được toàn bộ các viễn cảnh có thể. Tuy vậy, một điều đáng ngạc nhiên là phần lớn các hệ CSDL sử dụng LRU bất chấp các khuyết điểm của chiến lược đó.

Chiến lược được sử dụng bởi bộ quản trị buffer để thay thế khối bị ảnh hưởng bởi các nhân tố khác hơn là nhân tố thời gian tại đó khối được tham khảo trở lại. Nếu hệ thống đang xử lý các yêu cầu của một vài người sử dụng cạnh tranh, hệ thống (con) điều khiển cạnh tranh (concurrency-control subsystem) có thể phải làm trễ một số yêu cầu để đảm bảo tính nhất quán của CSDL. Nếu bộ quản trị buffer được cho các thông tin từ hệ thống điều khiển cạnh tranh mà nó nêu rõ những yêu cầu nào đang bị làm trễ, nó có thể sử dụng các thông tin này để thay đổi chiến lược thay thế khối của nó. Đặc biệt, các khối cần thiết bởi các yêu cầu tích cực (active requests) có thể được giữ lại trong buffer, toàn bộ các bất lợi đổ dồn lên các khối cần thiết bởi các yêu cầu bị làm trễ.

Hệ thống (con) khôi phục (crash-recovery subsystem) áp đặt các ràng buộc nghiêm nhặt lên việc thay thế khối. Nếu một khối bị sửa đổi, bộ quản trị buffer không được phép viết lại phiên bản mới của khối trong buffer lên đĩa, vì điều này phá huỷ phiên bản cũ. Thay vào đó, bộ quản trị khối phải tìm kiếm quyền từ hệ thống khôi phục trước khi viết khối. Hệ

thống khôi phục có thể đòi hỏi một số khối nhất định khác là xuất bắt buộc (forced output) trước khi cấp quyền cho bộ quản trị buffer để xuất ra khối được yêu cầu.

TỔ CHỨC FILE

Một file được tổ chức logic như một dãy các mẩu tin (record). Các mẩu tin này được ánh xạ lên các khối đĩa. File được cung cấp như một xây dựng cơ sở trong hệ điều hành, như vậy ta sẽ giả thiết sự tồn tại của hệ thống file nền. Ta cần phải xét những phương pháp biểu diễn các mô hình dữ liệu logic trong thuật ngữ file.

Các khối có kích cỡ cố định được xác định bởi tính chất vật lý của đĩa và bởi hệ điều hành, song kích cỡ của mẩu tin lại thay đổi. Trong CSDL quan hệ, các bộ của các quan hệ khác nhau nói chung có kích cỡ khác nhau.

Một tiếp cận để ánh xạ một CSDL đến các file là sử dụng một số file, và lưu trữ các mẩu tin thuộc chỉ một độ dài cố định vào một file đã cho nào đó. Một cách khác là cấu trúc các file sao cho ta có thể điều tiết nhiều độ dài cho các mẩu tin. Các file của các mẩu tin độ dài cố định dễ dàng thực thi hơn file của các mẩu tin độ dài thay đổi.

MẨU TIN ĐỘ DÀI CỐ ĐỊNH (Fixed-Length Records)

Xét một file các mẩu tin account đối với CSDL ngân hàng, mỗi mẩu tin của file này được xác định như sau:

type

depositor = record

branch_name: char(20);

account_number: char(10);

balance:real;

end

Giả sử mỗi một ký tự chiếm 1 byte và mỗi số thực chiếm 8 byte, như vậy mẩu tin account có độ dài 40 bytes. Một cách tiếp cận đơn giản là sử dụng 40 byte đầu tiên cho mẩu tin thứ nhất, 40 byte kế tiếp cho mẩu tin thứ hai, ... Cách tiếp cận đơn giản này nảy sinh những vấn đề sau;

0	Perryridge	A-102	400
1	Round Hill	A-305	350
2	Mianus	A-215	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600
8	Perryridge	A-218	700

1. File F chứa các mẩu tin account

0	Perryridge	A-102	400
1	Round Hill	A-305	350
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600
8	Perryridge	A-218	700

2. File F sau khi xóa mẩu tin 2 và di chuyển các mẩu tin sau nó

0	Perryridge	A-102	400
1	Round Hill	A-305	350
8	Perryridge	A-218	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600

3. File F sau khi xóa mẩu tin 2 và di chuyển mẩu tin cuối vào vị chí của nó

header				
0	Perryridge	A-102	400	
1				
2	Mianus	A-215	700	
3	Downtown	A-101	500	
4				
5	Perryridge	A-201	900	
6				
7	Downtown	A-110	600	
8	Perryridge	A-218	700	

4. File F sau khi xoá các mẩu tin 1, 4, và 6 với chiến lược bỏ ngỏ và danh sách tự do

1. Khó khăn khi xoá một mẩu tin từ cấu trúc này. Không gian bị chiếm bởi mẩu tin bị xoá phải được lấp đầy với mẩu tin khác của file hoặc ta phải đánh dấu mẩu tin bị xoá.
2. Trừ khi kích cỡ khối là bội của 40, nếu không một số mẩu tin sẽ bắt chéo qua biên khối, có nghĩa là một phần mẩu tin được lưu trong một khối, một phần khác được lưu trong một khối khác. như vậy đòi hỏi phải truy xuất hai khối để đọc/viết một mẩu tin "bắc cầu" đó.

Khi một mẩu tin bị xoá, ta có thể di chuyển mẩu tin kề sau nó vào không gian bị chiếm một cách hình thức bởi mẩu tin bị xoá, rồi mẩu tin kế tiếp vào không gian bị chiếm của mẩu tin vừa được di chuyển, cứ như vậy cho đến khi mỗi mẩu tin đi sau mẩu tin bị xoá được dịch chuyển hướng về đầu. Cách tiếp cận này đòi hỏi phải di chuyển một số lớn các mẩu tin. Một cách tiếp cận khác đơn giản hơn là di chuyển mẩu tin cuối cùng vào không gian bị chiếm bởi mẩu tin bị xoá. Song cách tiếp cận này đòi hỏi phải truy xuất khối bổ xung. Vì hoạt động xen xảy ra thường xuyên hơn hoạt động xoá, ta có thể chấp nhận việc để "ngỏ" không gian bị chiếm bởi mẩu tin bị xoá, và chờ một hoạt động xen đến sau để tái sử dụng không gian đó. Một dấu trên mẩu tin bị xoá là không đủ vì sẽ gây khó khăn cho việc tìm kiếm không gian "tự do" đó khi xen. Như vậy ta cần đưa vào cấu trúc bổ xung. ở đầu của file, ta cấp phát một số byte nhất định làm header của file. Header này sẽ chứa đựng thông tin về file. Header chứa địa chỉ của mẩu tin bị xoá thứ nhất, trong nội dung của mẩu tin này có chứa địa chỉ của mẩu tin bị xoá thứ hai và cứ như vậy. Như vậy, các mẩu tin bị xoá sẽ

tạo ra một danh sách liên kết được gọi là danh sách tự do (free list). Khi xen mẩu tin mới, ta sử dụng con trỏ đầu danh sách được chứa trong header để xác định danh sách, nếu danh sách không rỗng ta xen mẩu tin mới vào vùng được trỏ bởi con trỏ đầu danh sách nếu không ta xen mẩu tin mới vào cuối file.

Xen và xoá đối với file mẩu tin độ dài cố định thực hiện đơn giản vì không gian được giải phóng bởi mẩu tin bị xoá đúng bằng không gian cần thiết để xen một mẩu tin. Đối với file của các mẩu tin độ dài thay đổi vấn đề trở nên phức tạp hơn nhiều.

MẨU TIN ĐỘ DÀI THAY ĐỔI (Variable-Length Records)

Mẩu tin độ dài thay đổi trong CSDL do bởi:

- Việc lưu trữ nhiều kiểu mẩu tin trong một file
- Kiểu mẩu tin cho phép độ dài trường thay đổi
- Kiểu mẩu tin cho phép lặp lại các trường

Có nhiều kỹ thuật để thực hiện mẩu tin độ dài thay đổi. Để minh hoạ ta sẽ xét các biểu diễn khác nhau trên các mẩu tin độ dài thay đổi có định dạng sau:

Type account_list = record

branch_name: char(20);

account_info: array[1..   ] of record

account_number: char(10);

balance: real;

end;

end

Biểu diễn chuỗi byte (Byte-String Representation)

Một cách đơn giản để thực hiện các mẩu tin độ dài thay đổi là gắn một ký hiệu đặc biệt End-of-record ( ) vào cuối mỗi record. Khi đó, ta có thể lưu mỗi mẩu tin như một chuỗi byte liên tiếp. Thay vì sử dụng một ký hiệu đặc biệt ở cuối của mỗi mẩu tin, một phiên bản của biểu diễn chuỗi byte lưu trữ độ dài mẩu tin ở bắt đầu của mỗi mẩu tin.

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥				
2	Mianus	A-101	800	⊥				
3	Downtown	A-211	500	A-222	600	⊥		
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥				

Biểu diễn chuỗi byte của các mẩu tin độ dài thay đổi

Biểu diễn chuỗi byte có các bất lợi sau:

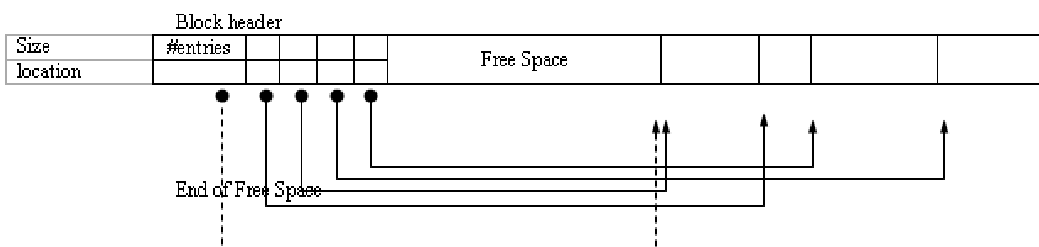
- Khó sử dụng không gian bị chiếm hình thức bởi một mẩu tin bị xoá, điều này dẫn đến một số lớn các mảnh nhỏ của lưu trữ đĩa bị lãng phí.
- Không có không gian cho sự phát triển các mẩu tin. Nếu một mẩu tin độ dài thay đổi dài ra, nó phải được di chuyển và sự di chuyển này là đắt giá nếu mẩu tin bị chốt.

Biểu diễn chuỗi byte không thường được sử dụng để thực hiện mẩu tin độ dài thay đổi, song một dạng sửa đổi của nó được gọi là cấu trúc khe-trang (slotted-page structure) thường được dùng để tổ chức mẩu tin trong một khối đơn.

Trong cấu trúc slotted-page, có một header ở bắt đầu của mỗi khối, chứa các thông tin sau:

- Số các đầu vào mẩu tin (record entries) trong header
- Điểm cuối không gian tự do (End of Free Space) trong khối
- Một mảng các đầu vào chứa vị trí và kích cỡ của mỗi mẩu tin

Các mẩu tin hiện hành được cấp phát kề nhau trong khối, bắt đàu từ cuối khối, Không gian tự do trong khối là một vùng kề nhau, nằm giữa đầu vào cuối cùng trong mảng header và mẩu tin đàu tiên. Khi một mẩu tin được xen vào, không gian cấp phát cho nó ở cuối của không gian tự do, và đầu vào tương ứng với nó được thêm vào header.



Nếu một mẩu tin bị xoá, không gian bị chiếm bởi nó được giải phóng, đầu vào ứng với nó được đặt là bị xoá (kích cỡ của nó được đặt chẳng hạn là -1). Sau đó, các mẩu tin trong khối trước mẩu tin bị xoá được di chuyển sao cho không gian tự do của khối lại là phần nằm giữa đầu vào cuối cùng của mảng header và mẩu tin đầu tiên. Con trỏ điểm cuối không gian tự do và các con trỏ ứng với mẩu tin bị di chuyển được cập nhật. Sự lớn lên

hay nhỏ đi của mẩu tin cũng sử dụng kỹ thuật tương tự (trong trường hợp khối còn không gian cho sự lớn lên của mẩu tin). Cái giá phải trả cho sự di chuyển không quá cao vì các khối có kích cỡ không lớn (thường 4Kbytes).

Biểu diễn độ dài cố định

Một cách khác để thực hiện mẩu tin độ dài thay đổi một cách hiệu quả trong một hệ thống file là sử dụng một hoặc một vài mẩu tin độ dài cố định để biểu diễn một mẩu tin độ dài thay đổi. Hai kỹ thuật thực hiện file của các mẩu tin độ dài thay đổi sử dụng mẩu tin độ dài cố định là:

1. Không gian dự trữ (reserved space). Giả thiết rằng các mẩu tin có độ dài không vượt quá một ngưỡng (độ dài tối đa). Ta có thể sử dụng mảu tin độ dài cố định (có độ dài tối đa), Phần không gian chưa dùng đến được lấp đầy bởi một ký tự đặc biệt: null hoặc End-of-record.
2. Contrỏ (Pointers). Mẩu tin độ dài thay đổi được biểu diễn bởi một danh sách các mẩu tin độ dài cố định, được "móc xích" với nhau bởi các con trỏ.

Sự bất lợi của cấu trúc con trỏ là lãng phí không gian trong tất cả các mẩu tin ngoại trừ mẩu tin đầu tiên trong danh sách (mẩu tin đầu tiên cần trường branch_name, các mẩu tin sau trong danh sách không cần thiết có trường này!). Để giải quyết vấn đề này người ta đề nghị phân các khối trong file thành hai loại:

Khối neo (Anchor block). chứa chỉ các mẩu tin đầu tiên trong danh sách

Khối tràn (Overflow block). chứa các mẩu tin còn lại của danh sách

Như vậy, tất cả các mẩu tin trong một khối có cùng độ dài, cho dù file có thể chứa các mẩu tin không cùng độ dài.

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥	⊥	⊥	⊥	⊥
2	Mianus	A-101	800	⊥	⊥	⊥	⊥	⊥
3	Downtown	A-211	500	A-222	600	⊥	⊥	⊥
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥

0	Perryridge	A-102	400	•
1		A-201	900	•
2		A-210	700	•
3	Round Hill	A-301	350	•
4	Mianus	A-101	800	•
5	Downtown	A-211	500	
6	Redwood	A-300	650	•
7	Brighton	A-111	750	•

Perryridge	A-102	400	•
Round Hill	A-301	350	•
Mianus	A-101	800	•
Downtown	A-211	500	•
Redwood	A-300	650	•
Brighton	A-111	750	•

1.1.1.1.1.1 Khối neo

A-	900	•
A-210	700	•

TỔ CHỨC CÁC MẪU TIN TRONG FILE

Ta đã xét làm thế nào để biểu diễn các mẩu tin trong một cấu trúc file. Một thể hiện của một quan hệ là một tập hợp các mẩu tin. Đã cho một tập hợp các mẩu tin, vấn đề đặt ra là làm thế nào để tổ chức chúng trong một file. Có một số cách tổ chức sau:

Tổ chức file đống (Heap File Organization). Trong tổ chức này, một mẩu tin bất kỳ có thể được lưu trữ ở bất kỳ nơi nào trong file, ở đó có không gian cho nó. Không có thứ tự nào giữa các mẩu tin. Một file cho một quan hệ.

Tổ chức file tuần tự (Sequential File Organization). Trong tổ chức này, các mẩu tin được lưu trữ thứ tự tuần tự, dựa trên giá trị của khoá tìm kiếm của mỗi mẩu tin.

Tổ chức file băm (Hashed File Organization). Trong tổ chức này, có một hàm băm được tính toán trên thuộc tính nào đó của mẩu tin. Kết quả của hàm băm xác định mẩu tin được bố trí trong khối nào trong file. Tổ chức này liên hệ chặt chẽ với cấu trúc chỉ mục.

Tổ chức file cụm (Clustering File Organization). Trong tổ chức này, các mẩu tin của một vài quan hệ khác nhau có thể được lưu trữ trong cùng một file. Các mẩu tin có liên hệ của các quan hệ khác nhau được lưu trữ trên cùng một khối sao cho một hoạt động I/O đem lại các mẩu tin có liên hệ từ tất cả các quan hệ.

TỔ CHỨC FILE TUẦN TỰ

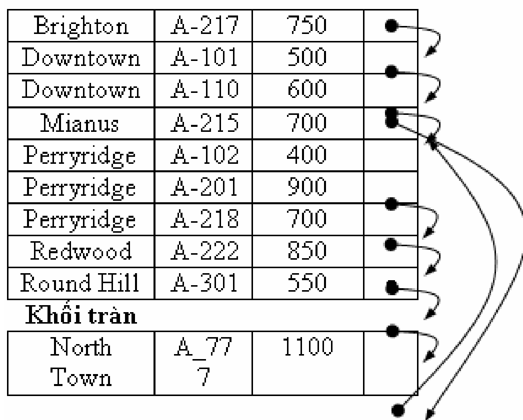
Tổ chức file tuần tự được thiết kế để xử lý hiệu quả các mẩu tin trong thứ tự được sắp dựa trên một khoá tìm kiếm (search key) nào đó. Để cho phép tìm lại nhanh chóng các mẩu tin theo thứ tự khoá tìm kiếm, ta "xích" các mẩu tin lại bởi các con trỏ. Con trỏ trong mỗi mẩu tin trỏ tới mẩu tin kế theo thứ tự khoá tìm kiếm. Hơn nữa, để tối ưu hoá số khối truy xuất trong xử lý file tuần tự, ta lưu trữ vật lý các mẩu tin theo thứ tự khoá tìm kiếm hoặc gần với khoá tìm kiếm như có thể.

Tổ chức file tuần tự cho phép đọc các mẩu tin theo thứ tự được sắp mà nó có thể hưu dụng cho mục đích trình bày cũng như cho các thuật toán xử lý vấn tin (query-processing algorithms).

Brighton	A-217	750	
Downtown	A-101	500	
Downtown	A-110	600	
Mianus	A-215	700	
Perryridge	A-102	400	
Perryridge	A-201	900	
Perryridge	A-218	700	

Khó khăn gặp phải của tổ chức này là việc duy trì thứ tự tuần tự vật lý của các mẩu tin khi xẩy ra các hoạt động xen, xoá, do cái giá phải trả cho việc di chuyển các mẩu tin khi xen, xoá. Ta có thể quản trị vấn đề xoá bởi dùng dây chuyền các con trỏ như đã trình bày trước đây. Đối với xen, ta có thể áp dụng các quy tắc sau:

1. Định vị mẩu tin trong file mà nó đi trước mẩu tin được xen theo thứ tự khoá tìm kiếm.
2. Nếu có mẩu tin tự do (không gian của mẩu tin bị xoá) trong cùng khối, xen mẩu tin vào khối này. Nếu không, xen mẩu tin mới vào một khối tràn. Trong cả hai trường hợp, điều chỉnh các con trỏ sao cho nó móc xích các mẩu tin theo thứ tự của khoá tìm kiếm.



TỔ CHỨC FILE CỤM

Nhiều hệ CSDL quan hệ, mỗi quan hệ được lưu trữ trong một file sao cho có thể lợi dụng được toàn bộ những cái mà hệ thống file của điều hành cung cấp. Thông thường, các bộ của một quan hệ được biểu diễn như các mẩu tin độ dài cố định. Như vậy các quan hệ có thể ánh xạ vào một cấu trúc file. Sự thực hiện đơn giản đó của một hệ CSDL quan hệ rất phù hợp với các hệ CSDL được thiết kế cho các máy tính cá nhân. Trong các hệ thống đó, kích cỡ của CSDL nhỏ. Hơn nữa, trong một số máy tính cá nhân, chủ yếu kích cỡ tổng thể mã đối tượng đối với hệ CSDL là nhỏ. Một cấu trúc file đơn giản làm suy giảm lượng mã cần thiết để thực thi hệ thống.

Cách tiếp cận đơn giản này, để thực hiện CSDL quan hệ, không còn phù hợp khi kích cỡ của CSDL tăng lên. Ta sẽ thấy những điểm lợi về mặt hiệu năng từ việc gán một cách thận trọng các mẩu tin với các khối, và từ việc tổ chức kỹ lưỡng chính bản thân các khối. Như vậy, có vẻ như là một cấu trúc file phức tạp hơn lại có lợi hơn, ngay cả trong trường hợp ta giữ nguyên chiến lược lưu trữ mỗi quan hệ trong một file riêng biệt.

Tuy nhiên, nhiều hệ CSDL quy mô lớn không nhờ cậy trực tiếp vào hệ điều hành nền để quản trị file. Thay vào đó, một file hệ điều hành được cấp phát cho hệ CSDL. Tất cả các quan hệ được lưu trữ trong một file này, và sự quản trị file này thuộc về hệ CSDL. Để thấy những điểm lợi của việc lưu trữ nhiều quan hệ trong cùng một file, ta xét vấn tin SQL sau:

SELECT account_number, customer_number, customer_treet, customer_city

FROM depositor, customer

WHEREdepositor.customer_name = customer.customername;

Câu vấn tin này tính một phép nối của các quan hệ depositor và customer. Như vậy, đối với mỗi bộ của depositor, hệ thống phải tìm bộ của customer có cùng giá trị customer_name. Một cách lý tưởng là việc tìm kiếm các mẩu tin này nhờ sự trợ giúp của chỉ mục. Bỏ qua việc tìm kiếm các mẩu tin như thế nào, ta chú ý vào việc truyền từ đĩa vào bộ nhớ. Trong trường hợp xấu nhất, mỗi mẩu tin ở trong một khối khác nhau, điều này buộc ta phải đọc một khối cho một mẩu tin được yêu cầu bởi câu vấn tin. Ta sẽ trình bày một cấu trúc file được thiết kế để thực hiện hiệu quả các câu vấn tin liên quan đến depositor customer. Các bộ depositor đối với mỗi customer_name được lưu trữ gần bộ customer có cùng customer_name. Cấu trúc này trộn các bộ của hai quan hệ với nhau, nhưng cho phép xử lý hiệu quả phép nối. Khi một bộ của của quan hệ customer được đọc, toàn bộ khối chứa bộ này được đọc từ đĩa vào trong bộ nhớ chính. Do các bộ tương ứng của depositor được lưu trữ trên đĩa gần bộ customer, khối chứa bộ customer chứa các bộ của quan hệ depositor cần cho xử lý câu vấn tin. Nếu một customer có nhiều account đến nỗi các mẩu tin depositor không lấp đầy trong một khối, các mẩu tin còn lại xuất hiện trong khối kế cận. Cấu trúc file này, được gọi là gom cụm (clustering), cho phép ta đọc nhiều mẩu tin được yêu cầu chỉ sử dụng một đọc khối, như vậy ta có thể xử lý câu vấn tin đặc biệt này hiệu quả hơn.

customer_name	customer_street	customer_city
Hays	Main	Brooklyn
Turner	Putnam	Stamford

Quan hệ

Hays	Main	Brooklyn	•
Hays	A-102		
Hays	A-220		
Hays	A-503		
Turner	Putnam	Stamford	•
Turner	A-305		

Cấu trúc file cụm với dây chuyền con trỏ

Hays	Main	Brooklyn
Hays	A-102	
Hays	A-220	
Hays	A-503	
Turner	Putnam	Stamford
Turner	A-305	

Cấu trúc file cụm

Tuy nhiên, cấu trúc gom cụm trên lại tỏ ra không có lợi bằng tổ chức lưu mỗi quan hệ trong một file riêng, đối với một số câu vấn tin, chẳng hạn:

SELECT *

FROM customer

Việc xác định khi nào thì gom cụm thường phụ thuộc vào kiểu câu vấn tin mà người thiết kế CSDL nghĩ rằng nó xẩy ra thường xuyên nhất. Sử dụng thận trọng gom cụm có thể cải thiện hiệu năng đáng kể trong việc xử lý câu vấn tin.

LƯU TRỮ TỰ ĐIỂN DỮ LIỆU

Một hệ CSDL cần thiết duy trì dữ liệu về các quan hệ, như sơ đồ của các quan hệ. Thông tin này được gọi là tự điển dữ liệu (data dictionary) hay mục lục hệ thống (system catalog). Trong các kiểu thông tin mà hệ thống phải lưu trữ là:

- Các tên của các quan hệ
- Các tên của các thuộc tính của mỗi quan hệ
- Các miền (giá trị) và các độ dài của các thuộc tính
- Các tên của các View được định nghĩa trên CSDL và định nghĩa của các view này
- Các ràng buộc toàn vẹn

Nhiều hệ thống còn lưu trữ các thông tin liên quan đến người sử dụng hệ thống:

- Tên của người sử dụng được phép
- Giải trình thông tin về người sử dụng

Các dữ liệu thống kê và mô tả về các quan hệ có thể cũng được lưu trữ:

- Số bộ trong mỗi quan hệ
- Phương pháp lưu trữ được sử dụng cho mỗi quan hệ (cụm hay không)

Các thông tin về mỗi chỉ mục trên mỗi quan hệ cũng cần được lưu trữ :

- Tên của chỉ mục
- Tên của quan hệ được chỉ mục
- Các thuộc tính trên nó chỉ mục được định nghĩa
- Kiểu của chỉ mục được tạo

Toàn bộ các thông tin này trong thực tế bao hàm một CSDL nhỏ. Một số hệ CSDL sử dụng những cấu trúc dữ liệu và mã mục đích đặc biệt để lưu trữ các thông tin này. Nói chung, lưu trữ dữ liệu về CSDL trong chính CSDL vẫn được ưa chuộng hơn. Bằng cách sử dụng CSDL để lưu trữ dữ liệu hệ thống, ta đơn giản hoá cấu trúc tổng thể của hệ thống và cho phép sử dụng đầy đủ sức mạnh của CSDL trong việc truy xuất nhanh đến dữ liệu hệ thống.

Sự chọn lựa chính xác biểu diễn dữ liệu hệ thống sử dụng các quan hệ như thế nào là do người thiết kế hệ thống quyết định. Như một ví dụ, ta đề nghị sự biểu diễn sau:

System_catalog_schema = (relation_name, number_of_attributes)

Attribute_schema = (attribute_name, relation_name, domain_type, position, length)

User_schema = (user_name, encrypted_password, group)

Index_schema = (index_name, relation_name, index_type, index_attributes)

View_schema = (view_name, definition)

CHỈ MỤC

Ta xét hoạt động tìm sách trong một thư viện. Ví dụ ta muốn tìm một cuốn sách của một tác giả nào đó. Đầu tiên ta tra trong mục lục tác giả, một tấm thẻ trong mục lục này sẽ chỉ cho ta biết có thể tìm thấy cuốn sách đó ở đâu. Các thẻ trong một mục lục được thư viện sắp xếp thứ tự theo vần chữ cái , như vậy giúp ta có thể tìm đến thẻ cần tìm nhanh chóng không cần phải duyệt qua tất cả các thẻ. Chỉ mục của một file trong các công việc hệ thống rất giống với một mục lục trong một thư viện. Tuy nhiên, chỉ mục được làm như mục lục được mô tả như trên, trong thực tế, sẽ quá lớn để được quản lý một cách hiệu quả. Thay vào đó, người ta sử dụng các kỹ thuật chỉ mục tinh tế hơn. Có hai kiểu chỉ mục:

- Chỉ mục được sắp (Ordered indices). được dựa trên một thứ tự sắp xếp theo các giá trị
- Chỉ mục băm (Hash indices). được dựa trên các giá trị được phân phối đều qua các bucket. Bucket mà một giá trị được gán với nó được xác định bởi một hàm, được gọi là hàm băm (hash function)

Đối với cả hai kiểu này, ta sẽ nêu ra một vài kỹ thuật, đáng lưu ý là không kỹ thuật nào là tốt nhất. Mỗi kỹ thuật phù hợp với các ứng dụng CSDL riêng biệt. Mỗi kỹ thuật phải được đánh giá trên cơ sở của các nhân tố sau:

- Kiểu truy xuất: Các kiểu truy xuất được hỗ trợ hiệu quả. Các kiểu này bao hàm cả tìm kiếm mẩu tin với một giá trị thuộc tính cụ thể hoặc tìm các mẩu tin với giá trị thuộc tính nằm trong một khoảng xác định.
- Thời gian truy xuất: Thời gian để tìm kiếm một hạng mục dữ liệu hay một tập các hạng mục.
- Thời gian xen: Thời gian để xen một hạng mục dữ liệu mới. giá trị này bao hàm thời gian để tìm vị trí xen thích hợp và thời gian cập nhật cấu trúc chỉ mục.
- Thời gian xoá: Thời gian để xoá một hạng mục dữ liệu. giá trị này bao hàm thời gian tìm kiếm hạng mục cần xoá, thời gian cập nhật cấu trúc chỉ mục.
- Tổng phí tổn không gian: Không gian phụ bị chiếm bởi một cấu trúc chỉ mục.

Một file thường đi kèm với một vài chỉ mục. Thuộc tính hoặc tập hợp các thuộc tính được dùng để tìm kiếm mẩu tin trong một file được gọi là khoá tìm kiếm. Chú ý rằng định nghĩa này khác với định nghĩa khoá sơ cấp (primary key), khoá dự tuyển (candidate key), và siêu khoá (superkey). Như vậy, nếu có một vài chỉ mục trên một file, có một vài khoá tìm kiếm tương ứng.

CHỈ MỤC ĐƯỢC SẮP.

Một chỉ mục lưu trữ các giá trị khoá tìm kiếm trong thứ tự được sắp, và kết hợp với mỗi khoá tìm kiếm, các mẩu tin chứa khoá tìm kiếm này. Các mẩu tin trong file được chỉ mục có thể chính nó cũng được sắp. Một file có thể có một vài chỉ mục trên những khoá tìm kiếm khác nhau. Nếu file chứa các mẩu tin được sắp tuần tự, chỉ mục trên khoá tìm kiếm xác định thứ tự này của file được gọi chỉ mục sơ cấp (primary index). Các chỉ mục sơ cấp cũng được gọi là chỉ mục cụm (clustering index). Khoá tìm kiếm của chỉ mục sơ cấp thường là khoá sơ cấp (khoá chính). Các chỉ mục, khoá tìm kiếm của nó xác định một thứ tự khác với thứ tự của file, được gọi là các chỉ mục thứ cấp (secondary indices) hay các chỉ mục không cụm (nonclustering indices).

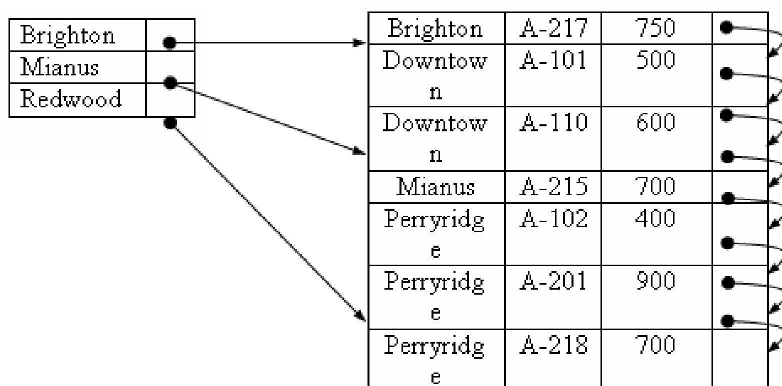
Brighton	A-217	750	•
Downtown	A-101	500	•
Downtown	A-110	600	•
Mianus	A-215	700	•
Perryridge	A-102	400	•
Perryridge	A-201	900	•
Perryridge	A-218	700	
Redwood	A-222	850	

Chỉ mục sơ cấp.

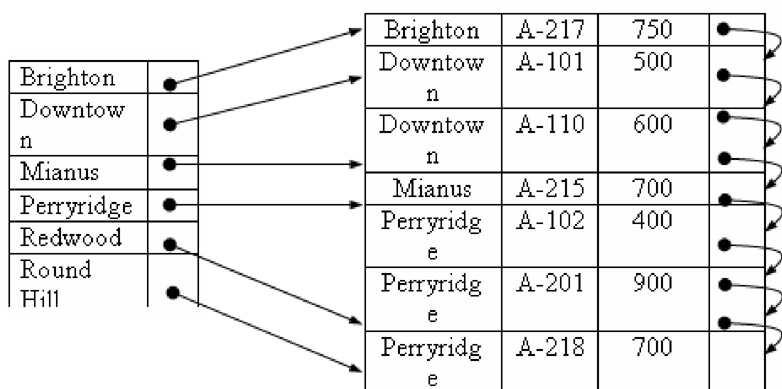
Trong phần này, ta giả thiết rằng tất cả các file được sắp thứ tự tuần tự trên một khoá tìm kiếm nào đó. Các file như vậy, với một chỉ mục sơ cấp trên khoá tìm kiếm này, được gọi là file tuần tự chỉ mục (index-sequential files). Chúng biểu diễn một trong các sơ đồ xưa nhất được dùng trong hệ CSDL. Chúng được thiết kế cho các ứng dụng đòi hỏi cả xử lý tuần tự toàn bộ file lẫn truy xuất ngẫu nhiên đến một mẩu tin.

Chỉ mục đặc và chỉ mục thưa (Dense and Sparse Indices)

Chỉ mục đặc



Chỉ mục thưa



Có hai loại chỉ mục được sắp:

- Chỉ mục đặc. Mỗi mẩu tin chỉ mục (đầu vào chỉ mục/ index entry) xuất hiện đối với mỗi giá trị khoá tìm kiếm trong file. mẩu tin chỉ mục chứa giá trị khoá tìm kiếm và một con trỏ tới mẩu tin dữ liệu đầu tiên với giá trị khoá tìm kiếm đó.
- Chỉ mục thưa. Một mẩu tin chỉ mục được tạo ra chỉ với một số giá trị. Cũng như với chỉ mục đặc, mỗi mẩu tin chỉ mục chứa một giá trị khoá tìm kiếm và một con trỏ tới mẩu tin dữ liệu đầu tiên với giá trị khoá tìm kiếm này. Để định vị một mẩu tin, ta tìm đầu vào chỉ mục với giá trị khoá tìm kiếm lớn nhất trong các giá trị khoá tìm kiếm nhỏ hơn hoặc bằng giá trị khoá tìm kiếm đang tìm. Ta bắt đầu từ mẩu tin được trỏ tới bởi đầu vào chỉ mục, và lần theo các con trỏ trong file (dữ liệu) đến tận khi tìm thấy mẩu tin mong muốn.

Ví dụ: Giả sử ta tìm các kiếm mẩu tin đối với chi nhánh Perryridge, sử dụng chỉ mục đặc. Đầu tiên, tìm Perryridge trong chỉ mục (tìm nhị phân!), đi theo con trỏ tương ứng đến mẩu

tin dữ liệu (với Branch_name = Perryridge) đầu tiên, xử lý mẩu tin này, sau đó đi theo con trỏ trong mẩu tin này để định vị mẩu tin kế trong thứ tự khoá tìm kiếm, xử lý mẩu tin này, tiếp tục như vậy đến tận khi đạt tới mẩu tin có Branch_name khác với Perryridge.

Đối với chỉ mục thưa, đầu tiên tìm trong chỉ mục, đầu vào có Branch_name lớn nhất trong các đầu vào có Branch_name nhỏ hơn hoặc bằng Perryridge, ta tìm được đầu vào với Mianus, lần theo con trỏ tương ứng đến mẩu tin dữ liệu, đi theo con trỏ trong mẩu tin Mianus để định vị mẩu tin kế trong thứ tự khoá tìm kiếm và cứ như vậy đến tận khi đạt tới mẩu tin dữ liệu Perryridge đầu tiên, sau đó xử lý bắt đầu từ điểm này.

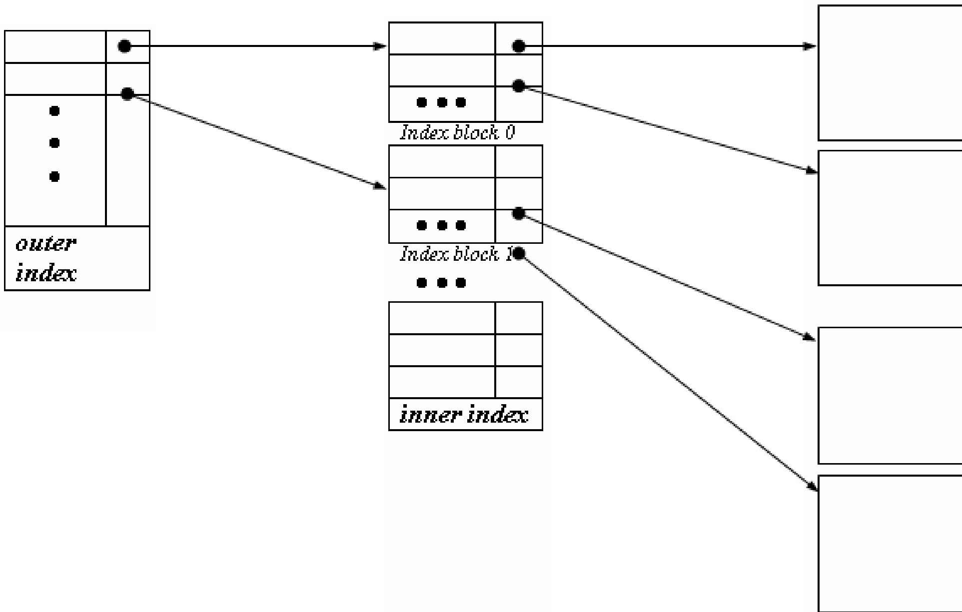
Chỉ mục đặc cho phép tìm kiếm mẩu tin nhanh hơn chỉ mục thưa, song chỉ mục thưa lại đòi hỏi ít không gian hơn chỉ mục đặc. Hơn nữa, chỉ mục thưa yêu cầu một tổn phí duy trì nhỏ hơn đối với các hoạt động xen, xoá.

Người thiết kế hệ thống phải cân nhắc sự cân đối giữa thời truy xuất và tổn phí không gian. Một thoả hiệp tốt là có một chỉ mục thưa với một đầu vào chỉ mục cho mỗi khối, vì như vậy cái giá nổi trội trong xử lý một yêu cầu CSDL là thời gian mang một khối từ đĩa vào bộ nhớ chính. Mỗi khi một khối được mang vào, thời gian quét toàn bộ khối là không đáng kể. Sử dụng chỉ mục thưa, ta tìm khối chứa mẩu tin cần tìm. Như vậy, trừ phi mẩu tin nằm trên khối tràn, ta tối thiểu hoá được truy xuất khối, trong khi giữ được kích cỡ của chỉ mục nhỏ như có thể.

Chỉ mục nhiều mức

Chỉ mục có thể rất lớn, ngay cả khi sử dụng chỉ mục thưa, và không thể chứa đủ trong bộ nhớ một lần. Tìm kiếm đầu vào chỉ mục đối với các chỉ mục như vậy đòi hỏi phải đọc vài khối đĩa. Tìm kiếm nhị phân có thể được sử dụng để tìm một đầu vào trên file chỉ mục, song vẫn phải truy xuất khoảng logB khối, với B là số khối đĩa chứa chỉ mục. Nếu B lớn, thời gian truy xuất này là đáng kể! Hơn nữa nếu sử dụng các khối tràn, tìm kiếm nhị phân không sử dụng được và như vậy việc tìm kiếm phải làm tuần tự. Nó đòi hỏi truy xuất lên đến B khối!!

Để giải quyết vấn đề này, Ta xem file chỉ mục như một file tuần tự và xây dựng chỉ mục thưa cho nó. Để tìm đầu vào chỉ mục, ta tìm kiếm nhị phân trên chỉ mục "ngoài" để được mẩu tin có khoá tìm kiếm lớn nhất trong các mẩu tin có khoá tìm kiếm nhỏ hơn hoặc bằng khoá muốn tìm. Con trỏ tương ứng trỏ tới khối của chỉ mục "trong". Trong khối này, tìm kiếm mẩu tin có khoá tìm kiếm lớn nhất trong các mẩu tin có khoá tìm kiếm nhỏ hơn hoặc bằng khoá muốn tìm, trường con trỏ của mẩu tin này trỏ đến khối chứa mẩu tin cần tìm. Vì chỉ mục ngoài nhỏ, có thể nằm sẵn trong bộ nhớ chính, nên một lần tìm kiếm chỉ cần một truy xuất khối chỉ mục. Ta có thể lặp lại quá trình xây dựng trên nhiều lần khi cần thiết. Chỉ mục với không ít hơn hai mức được gọi là chỉ mục nhiều mức. Với chỉ mục nhiều mức, việc tìm kiếm mẩu tin đòi hỏi truy xuất khối ít hơn đáng kể so với tìm kiếm nhị phân.



Cập nhật chỉ mục

Mỗi khi xen hoặc xoá một mẩu tin, bắt buộc phải cập nhật các chỉ mục kèm với file chứa mẩu tin này. Dưới đây, ta mô tả các thuật toán cập nhật cho các chỉ mục một mức

- Xoá. Để xoá một mẩu tin, đầu tiên phải tìm mẩu tin muốn xoá. Nếu mẩu tin bị xoá là mẩu tin đầu tiên trong dây chuyền các mẩu tin được xác định bởi con trỏ của đầu vào chỉ mục trong quá trình tìm kiếm, có hai trường hợp phải xét: nếu mẩu tin bị xoá là mẩu tin duy nhất trong dây chuyền, ta xoá đầu vào trong chỉ mục tương ứng, nếu không, ta thay thế khoá tìm kiếm trong đầu vào chỉ mục bởi khoá tìm kiếm của mẩu tin kề sau mẩu tin bị xoá trong dây chuyền, con trỏ bởi địa chỉ mẩu tin kề sau đó. Trong trường hợp khác, việc xoá mẩu tin không dẫn đến việc điều chỉnh chỉ mục.
- Xen. Trước tiên, tìm kiếm dựa trên khoá tìm kiếm của mẩu tin được xen. Nếu là chỉ mục đặc và giá trị khoá tìm kiếm không xuất hiện trong chỉ mục, xen giá trị khoá này và con trỏ tới mẩu tin vào chỉ mục. Nếu là chỉ mục thưa và lưu đầu vào cho mỗi khối, không cần thiết phải thay đổi trừ phi khối mới được tạo ra. Trong trường hợp đó, giá trị khoá tìm kiếm đầu tiên trong khối mới được xen vào chỉ mục.

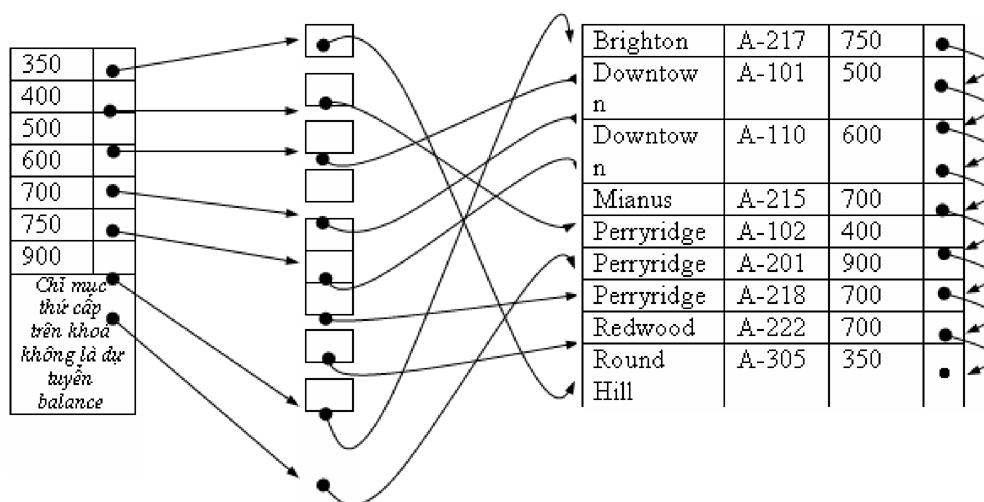
Giả thuật xen và xoá đối với chỉ mục nhiều mức là một mở rộng đơn giản của các giả thuật vừa được mô tả.

Chỉ mục thứ cấp.

Chỉ mục thứ cấp trên một khoá dự tuyển giống như chỉ mục sơ cấp đặc ngoại trừ các mẩu tin được trỏ đến bởi các giá trị liên tiếp trong chỉ mục không được lưu trữ tuần tự. Nói

chung, chỉ mục thứ cấp có thể được cấu trúc khác với chỉ mục sơ cấp. Nếu khoá tìm kiếm của chỉ mục sơ cấp không là khoá dự tuyển, chỉ mục chỉ cần trỏ đến mẩu tin đầu tiên với một giá trị khoá tìm kiếm riêng là đủ (các mẩu tin khác cùng giá trị khoá này có thể tìm lại được nhờ quét tuần tự file).

Nếu khoá tìm kiếm của một chỉ mục thứ cấp không là khoá dự tuyển, việc trỏ tới mẩu tin đầu tiên với giá trị khoá tìm kiếm riêng không đủ, do các mẩu tin trong file không còn được sắp tuần tự theo khoá tìm kiếm của chỉ mục thứ cấp, chúng có thể nằm ở bất kỳ vị trí nào trong file. Bởi vậy, chỉ mục thứ cấp phải chứa tất cả các co trỏ tới mỗi mẩu tin. Ta có thể sử dụng mức phụ gián tiếp để thực hiện chỉ mục thứ cấp trên các khoá tìm kiếm không là khoá dự tuyển. Các con trỏ trong chỉ mục thứ cấp như vậy không trực tiếp trỏ tới mẩu tin mà trỏ tới một bucket chứa các con trỏ tới file.



Chỉ mục thứ cấp phải là đặc, với một đầu vào chỉ mục cho mỗi mẩu tin. Chỉ mục thứ cấp cải thiện hiệu năng các vấn tin sử dụng khoá tìm kiếm không là khóa của chỉ mục sơ cấp, tuy nhiên nó lại đem lại một tổn phí sửa đổi CSDL đáng kể.Việc quyết định các chỉ mục thứ cấp nào là cần thiết dựa trên đánh giá của nhà thiết kế CSDL về tần xuất vấn tin và sửa đổi.

FILE CHỈ MỤC B+-CÂY (B+-Tree Index file)

Tổ chức file chỉ mục tuần tự có một nhược điếm chính là làm giảm hiệu năng khi file lớn lên. Để khắc phục nhược điểm đó đòi hỏi phải tổ chức lại file. Cấu trúc chỉ mục B+-cây là cấu trúc được sử dụng rộng rãi nhất trong các cấu trúc đảm bảo được tính hiệu quả của chúng bất chấp các hoạt động xen, xoá. Chỉ mục B+-cây là một dạng cây cân bằng (mọi đường dẫn từ gốc đến lá có cùng độ dài). Mỗi nút không là lá có số con nằm trong khoảng giữa m/2 và m, trong đó m là một số cố định được gọi là bậc của B+-cây. Ta thấy rằng

cấu trúc B+-cây cũng đòi hỏi một tổn phí hiệu năng trên xen và xoá cũng như trên không gian. Tuy nhiên, tổn phí này là chấp nhận được ngay cả đối với các file có tần suất sửa đổi cao.

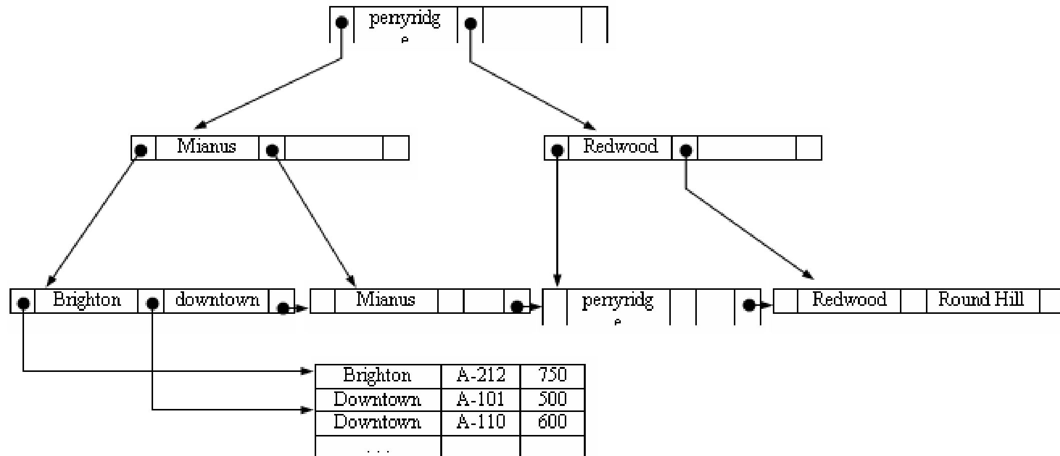
Cấu trúc của B+-cây

Một chỉ mục B+-cây là một chỉ mục nhiều mức, nhưng có cấu trúc khác với file tuần tự chỉ mục nhiều mức (multilevel index-sequential). Một nút tiêu biểu của B+-cây chứa đến n-1 giá trị khoá tìm kiếm. K1, K2, ..., Kn-1, và n con trỏ P1, P2, ..., Pn, các giá trị khoá trong nút được sắp thứ tự: i < j Ki < Kj.

P1	K1	P2	K2	. . .	Pn-1	Kn-1	Pn

Trước tiên, ta xét cấu trúc của nút lá. Đối với i = 1, 2, ..., n-1, con trỏ Pi trỏ tới hoặc mẩu tin với giá trị khoá Ki hoặc tới một bucket các con trỏ mà mỗi một trong chúng trỏ tới một mẩu tin với

giá trị khoá Ki. Cấu trúc bucket chỉ được sử dụng trong các trường hợp: hoặc khoá tìm kiếm không là khoá sơ cấp hoặc file không được sắp theo khoá tìm kiếm. Con trỏ Pn được dùng vào mục đích đặc biệt: Pn được dùng để móc xích các nút lá lại theo thứ tự khoá tìm kiếm, điều này cho phép xử lý tuần tự file hiệu quả. Bây giờ ta xem các giá trị khoá tìm kiếm được gắn với một nút lá như thế nào. Mỗi nút nút lá có thể chứa đến n-1 giá trị. Khoảng giá trị mà mỗi nút lá chứa là không chồng chéo. Như vậy, nếu Li và Lj là hai nút lá với i < j thì mỗi giá trị khoá trong nút Li nhỏ hơn mọi giá trị khoá trong Lj . Nếu chỉ mục B+-cây là đặc, mỗi giá trị khoá tìm kiếm phải xuất hiện trong một nút lá nào đó.



Các nút không là lá của một B+-cây tạo ra một chỉ mục nhiều mức trên các nút lá. Cấu trúc của các nút không là lá tương tự như cấu trúc nút lá ngoại trừ tất cả các con trỏ đều trỏ đến các nút của cây. Các nút không là lá có thể chứa đến m con trỏ và phải chứa không ít hơn m/2 con trỏ ngoại trừ nút gốc. Nút gốc được phép chứa ít nhất 2 con trỏ. Số con trỏ trong một nút được gọi là số nan (fanout) của nút.

Con trỏ Pi của một nút không là lá (chứa p con trỏ, 1 < i < p) trỏ đến một cây con chứa các giá trị khoá tìm kiếm nhỏ hơn Ki và lớn hơn hoặc bằng Ki-1. Con trỏ P1 trỏ đến cây con chứa các giá trị khoá tìm kiếm nhỏ hơn K1. Con trỏ Pp trỏ tới cây con chứa các khoá tìm kiếm lớn hơn Kp-1.

Các vấn tin trên B+-cây

Ta xét xử lý vấn tin sử dụng B+-cây như thế nào ? Giả sử ta muốn tìn tất cả các mẩu tin với giá trị khoá tìm kiếm k. Đầu tiên, ta kiểm tra nút gốc, tìm giá trị khoá tìm kiếm nhỏ nhất lớn hơn k, giả sử giá trị khoá đó là Ki. Đi theo con trỏ Pi để di tới một nút khác. Nếu nút có p con trỏ và k > Kp-1, đi theo con trỏ Pp. Đến một nút tới, lặp lại quá trình tìm kiếm giá trị khoá tìm kiếm nhỏ nhất lớn hơn k và theo con trỏ tương ứng để đi tới một nút khác và tiếp tục như vậy đến khi đạt tới một nút lá. Con trỏ tương ứng trong nút lá hướng ta tới mẩu tin/bucket mong muốn. Số khối truy xuất không vượt quá $\lceil \log_{\lceil m/2 \rceil} K \rceil$, trong đó K là số giá trị khoá tìm kiếm trong B+-cây, m là bậc của cây.

Cập nhật trên B+-cây

- Xen. Sử dụng cùng kỹ thuật như tìm kiếm, ta tìm nút lá trong đó giá trị khoá tìm kiếm cần xen sẽ xuất hiện. Nếu khoá tìm kiếm đã xuất hiện rồi trong nút lá, xen mẩu tin vào trong file, thêm con trỏ tới mẩu tin vào trong bucket tương ứng. Nếu khoá tìm

kiếm chưa hiện diện trong nút lá, ta xen mẩu tin vào trong file rồi xen giá trị khoá tìm kiếm vào trong nút lá ở vị trí đúng (bảo tồn tính thứ tự), tạo một bucket mới với con trỏ tương ứng. Nếu nút lá không còn chỗ cho giá trị khoá mới, Một khối mới được yêu cầu từ hệ điều hành, các giá trị khoá trong nút lá được tách một nửa cho nút mới, giá trị khoá mới được xen vào vị trí đúng của nó vào một trong hai khối này. Điều này kéo theo việc xen giá trị khoá đầu khối mới và con trỏ tới khối mới vào nút cha. Việc xen cặp giá trị khoá và con trỏ vào nút cha này lại có thể dẫn đến việc tách nút ra làm hai. Quá trình này có thể dẫn đến tận nút gốc. Trong trường hợp nút gốc bị tách làm hai, một nút gốc mới được tạo ra và hai con của nó là hai nút được tách ra từ nút gốc cũ, chiều cao cây tăng lên một.

Procedure Insert(value V, pointer P)

Tìm nút lá L sẽ chứa giá trị V

Insert_entry(L, V, P)

end procedure

Procedure Insert_entry(node L, value V, pointer P)

If (L có không gian cho (V, P) then

Xen (V, P) vào L

elsebegin/* tách L */

Tạo nút L'

If (L là nút lá) then begin

V' là giá trị sao cho m/2 giá trị trong các giá trị L.K1, L.K2, ..., L.Km-1, V nhỏ hơn V'

n là chỉ số nhỏ nhất sao cho L.Kn V'

Di chuyển L.Pn, L.Kn, ..., L.Pm-1, L.Kn-1 sang L'

If (V < V') then xen (V, P) vào trong L else xen (P, V) vào trong L'

end else begin

V' là giá trị sao cho m/2 giá trị trong các giá trị L.K1, L.K2, ..., L.Km-1, V lớn hơn hoặc bằng V'

n là chỉ số nhỏ nhất sao cho L.Kn V'

Thêm Nil, L.Kn, L.Pn+1, L.Kn+1, ..., L.Pm-1, L.Km-1, L.Pm vào L'

Xoá L.Kn, L.Pn+1, L.Kn+1, ..., L.Pm-1, L.Km-1, L.Pm khỏi L

If (V < V') then xen (P, V) vào trong L else xen (P, V) vào trong L'

xoá (Nil, V') khỏi L'

end

If (L không là nút gốc) then Insert_entry(parent(L), V', L')

else begin

Tạo ra nút mới R với các nút con là L và L' với giá trị duy nhất trong nó là V'

Tạo R là gốc của cây

end

If (L) là một nút lá then begin

đặt L'.Pm = L.Pm

đặt L.Pm = L'

end

end

end procedure

- Xoá. Sử dụng kỹ thuật tìm kiếm tìm mẩu tin cần xoá, xoá nó khỏi file, xoá giá trị khoá tìm kiếm khỏi nút lá trong B+-cây nếu không có bucket kết hợp với giá trị khoá tìm kiếm hoặc bucket trở nên rỗng sau khi xoá con trỏ tương ứng trong nó. Việc xoá một giá trị khoá khỏi một nút của B+-cây có thể dẫn đến nút lá trở nên rỗng, phải trả lại, từ đó nút cha của nó có thể có số con nhỏ hơn ngưỡng cho phép, trong trường hợp đó hoặc phải chuyển một con từ nút anh em của nút cha đó sang nút cha nếu điều đó có thể (nút anh em của nút cha này còn số con m/2 sau khi chuyển đi một con). Nếu không, phải gom nút cha này với một nút anh em của nó, điều này dẫn tới xoá một nút trong khỏi cây, rồi xoá khỏi nút cha của nó một hạng, ... quá trình này có thể dẫn đến tận gốc. Trong trường hợp nút gốc chỉ còn một con sau xoá, cây phải thay nút gốc cũ bởi nút con của nó, nút gốc cũ phải trả lại cho hệ thống, chiều cao cây giảm đi một.

Procedure delete(value V, pointer P)

Tìm nút lá chứa (V, P)

delete_entry(L, V, P)

end procedure

Procedure delete_entry(node L, value V, pointer P)

xoá (V, P) khỏi L

If (L là nút gốc and L chỉ còn lại một con) then

Lấy con của L làm nút gốc mới của cây, xoá L

else If (L có quá ít giá trị/ con trỏ) then begin

L' là anh em kề trái hoặc phải của L

V' là giá trị ở giữa hai con trỏ L, L' (trong nút parent(L))

If (các đầu vào của L và L' có thể lấp đầy trong một khối) then begin

If (L là nút trước của L') then wsap_variables(L, L')

If (L không là lá) then nối V' và tất cả con trỏ, giá trị trong L với L'

else begin nối tất cả các cặp (K, P) trong L với L'; L'.Pp = L.Pp end

delete_entry(parent(L), V', L); xoá nút L

end

else begin

If (L' là nút trước của L) then begin

If (L không là nút lá) then begin

p là chỉ số sao cho L'.Pp là con trỏ cuối trong L'

xoá (L'.Kp-1, L'.Pp) khỏi L'

xen (L'.Pp, V') như phần tử đầu tiên trong L (right_shift tất cả các phần tử của L)

thay thế V' trong parent(L) bởi L'.Kp-1

end else begin

p là chỉ số sao cho L'.Pp là con trỏ cuối trong L'

xoá (L'.Pp, L'.Kp) khỏi L'

xen (L'.Pp, L'.Kp) như phần tử đầu tiên trong L (right_shift tất cả các phần tử của L)

thay thế V' trong parent(L) bởi L'.Kp

end

end < đối xứng với trường hợp then >

end

end procedure

Tổ chức file B+-cây

Trong tổ chức file B+-cây, các nút lá của cây lưu trữ các mẩu tin, thay cho các con trỏ tới file. Vì mẩu tin thường lớn hơn con trỏ, số tối đa các mẩu tin được lưu trữ trong một khối lá ít hơn số con trỏ trong một nút không lá. Các nút lá vẫn được yêu cầu được lấp đầy ít nhất là một nửa.

Xen và xoá trong tổ chức file B+-cây tương tự như trong chỉ mục B+-cây.

Khi B+-cây được sử dụng để tổ chức file, việc sử dụng không gian là đặc biệt quan trọng, vì không gian bị chiếm bởi mẩu tin là lớn hơn nhiều so với không gian bị chiếm bởi (khoá,con trỏ). Ta có thể cải tiến sự sử sụng không gian trong B+-cây bằng cách bao hàm nhiều nút anh em hơn khi tái phân phối trong khi tách và trộn. Khi xen, nếu một nút là đầy, ta thử phân phối lại một số đầu vào đến một trong các nút kề để tạo không gian cho đầu vào mới. Nếu việc thử này thất bại, ta mới thực hiện tách nút và phân chia các đầu vào giữa một trong các nút kề và hai nút nhận được do tách nút. Khi xoá, nếu nút chứa ít hơn 2m/3 đầu vào, ta thử mượn một đầu vào từ một trong hai nút anh em kề. Nếu cả hai đều có đúng 2m/3 mẩu tin, ta phân phối lại các đầu vào của nút cho hai nút anh em kề và xoá nút thứ 3. Nếu k nút được sử dụng trong tái phân phối (k-1 nút anh em), mỗi nút đảm bảo chứa ít nhất (k-1)m/k đầu vào. Tuy nhiên, cái giá phải trả cho cập nhật của cách tiếp cận này sẽ cao hơn.

FILE CHỈ MỤC B-CÂY (B-Tree Index Files)

Chỉ mục B-cây tương tự như chỉ mục B+-cây. Sự khác biệt là ở chỗ B-cây loại bỏ lưu trữ dư thừa các giá trị khoá tìm kiếm. Trong B-cây, các giá trị khoá chỉ xuất hiện một lần. Do các khoá tìm kiếm xuất hiện trong các nút không là lá không xuất hiện ở bất kỳ nơi nào khác nữa trong B-cây, ta phải thêm một trường con trỏ cho mỗi khoá tìm kiếm trong các nút không là lá. Con trỏ thêm vào này trỏ tới hoặc mẩu tin trong file hoặc bucket tương ứng.

Một nút lá B-cây tổng quát có dạng:

P1	K1	P2	K2	...	Pm-1	Km-1	Pm

Một nút không là lá có dạng:

P1	B1	K-1	P2	B2	K2	...	Pm-1	Bm-1	Km-1	Pm

Các con trỏ Pi là các con trỏ cây và được dùng như trong B+-cây. Các con trỏ Bi trong các nút không là lá là các con trỏ mẩu tin hoặc con trỏ bucket. Rõ ràng là số giá trị khoá trong nút không lá nhỏ hơn số giá trị trong nút lá. Số nút được truy xuất trong quá trình tìm kiếm trong một B-cây phụ thuộc nơi khoá tìm kiếm được định vị.

Xoá trong một B-cây phức tạp hơn trong một B+-cây. Xoá một đầu vào xuất hiện ở một nút không là lá kéo theo việc tuyển chọn một giá trị thích hợp trong cây con của nút chứa đầu vào bị xoá. Nếu khoá Ki bị xoá, khoá nhỏ nhất trong cây con được trỏ bởi Pi+1 phải được di chuyển vào vị trí của Ki. Nếu nút lá còn lại quá ít đầu vào, cần thiết các hoạt động bổ xung.

Định nghĩa chỉ mục trong SQL

Một chỉ mục được tạo ra bởi lệnh CREATE INDEX với cú pháp

CREATE INDEX < index-name > ON < relation_name > (< attribute-list >)

attribute-list là danh sách các thuộc tính của quan hệ được dùng làm khoá tìm kiếm cho chỉ mục. Nếu muốn khai báo là khoá tìm kiếm là khoá dự tuyển, thêm vào từ khoá UNIQUE:

CREATE UNIQUE INDEX < index-name > ON < relation_name > (< attribute-list >)

attribute-list phải tạo thành một khoá dự tuyển, nếu không sẽ có một thông báo lỗi.

Bỏ đi một chỉ mục sử dụng lệnh DROP:

DROP INDEX < index-name >

BĂM (HASHING)

BĂM TĨNH (Static Hashing)

Bất lợi của tổ chức file tuần tự là ta phải truy xuất một cấu trúc chỉ mục để định vị dữ liệu, hoặc phải sử dụng tìm kiếm nhị phân, và kết quả là có nhiều hoạt động I/O. Tổ chức file dựa trên kỹ thuật băm cho phép ta tránh được truy xuất một cấu trúc chỉ mục. Băm cung cung cấp một phương pháp để xây dựng các chỉ mục.

Tổ chức file băm

Trong tổ chức file băm, ta nhận được địa chỉ của khối đĩa chứa một mẩu tin mong muốn bởi tính toán một hàm trên giá trị khoá tìm kiếm của mẩu tin. thuật ngữ bucket được dùng để chỉ một đơn vị lưu trữ. Một bucket kiểu mẫu là một khối đĩa, nhưng có thể được chọn nhỏ hơn hoặc lớn hơn một khối đĩa.

K ký hiệu tập tất cả các giá trị khoá tìm kiếm, B ký hiệu tập tất cả các địa chỉ bucket. Một hàm băm h là một hàm từ K vào B : h: K   B

Xen một mẩu tin với giá trị khoá K vào trong file: ta tính h(K). Giá trị của h(K) là địa chỉ của bucket sẽ chứa mẩu tin. Nếu có không gian trong bucket cho mẩu tin, mẩu tin được lưu trữ trong bucket.

Tìm kiếm một mẩu tin theo giá trị khoá K: đầu tiên tính h(K), ta tìm được bucket tương ứng. sau dó tìm trong bucket này mẩu tin với giá trị khoá K mong muốn.

Xoá mẩu tin với giá trị khoá K: tính h(K), tìm trong bucket tương ứng mẩu tin mong muốn, xoá nó khỏi bucket.

Hàm băm

Hàm băm xấu nhất là hàm ánh xạ tất cả các giá trị khoá vào cùng một bucket. Hàm băm lý tưởng là hàm phân phối đều các giá trị khoá vào các bucket, như vậy mỗi bucket chứa một số lượng mẩu tin như nhau. Ta muốn chọn một hàm băm thoả mãn các tiêu chuẩn sau:

- Phân phối đều: Mỗi bucket được gán cùng một số giá trị khoá tìm kiếm trong tập hợp tất cả các giá trị khoá có thể
- Phân phối ngẫu nhiên: Trong trường hợp trung bình, các bucket được gán một số lượng giá trị khoá tìm kiếm gần bằng nhau.

Các hàm băm phải được thiết kế thận trọng. Một hàm băm xấu có thể dẫn đến việc tìm kiếm chiếm một thời gian tỷ lệ với số khoá tìm kiếm trong file.

Điều khiển tràn bucket

Khi xen một mẩu tin, nếu bucket tương ứng còn chỗ, mẩu tin được xen vào bucket, nếu không sẽ xảy ra tràn bucket. Tràn bucket do các nguyên do sau:

- Các bucket không đủ. Số các bucket nB phải thoả mãn nB > nr / fr trong đó nr là tổng số mẩu tin sẽ lưu trữ, fr là số mẩu tin có thể lấp đầy trong một bucket.
- Sự lệch. Một vài bucket được gán cho một số lượng mẩu tin nhiều hơn các bucket khác, như vậy một bucket có thể tràn trong khi các bucket khác vẫn còn không gian. Tình huống này được gọi là sự lệch bucket. Sự lệch xảy ra do hai nguyên nhân:

1. Nhiều mẩu tin có cùng khoá tìm kiếm
2. Hàm băm được chọn phân phối các giá trị khoá không đều

Ta quản lý tràn bucket bằng cách dùng các bucket tràn. Nếu một mẩu tin phải được xen vào bucket B nhưng bucket B đã đầy, khi đó một bucket tràn sẽ được cấp cho B và mẩu tin được xen vào bucket tràn này. Nếu bucket tràn cũng đầy một bucket tràn mới lại được cấp và cứ như vậy. Tất cả các bucket tràn của một bucket được “móc xích” với nhau thành một danh sách liên kết. Việc điều khiển tràn dùng danh sách liên kết như vậy được gọi là dây chuyền tràn. Đối với dây chuyền tràn, thuật toán tìm kiếm thay đổi chú ít: trước tiên ta cũng tính giá trị hàm băm trên khoá tìm kiếm, ta được bucket B, kiểm tra các mẩu tin, trong bucket B và tất cả các bucket tràn tương ứng, có giá trị khoá khớp với giá trị tìm không.

Một cách điều khiển tràn bucket khác là: Khi cần xen một mẩu tin vào một bucket nhưng nó đã đầy, thay vì cấp thêm một bucket tràn, ta sử dụng một hàm băm kế trong một dãy các hàm băm được chọn để tìm bucket khác cho mẩu tin, nếu bucket sau cũng đầy, ta lại sử dụng một hàm băm kế và cứ như vậy... Dãy các hàm băm thường được sử dụng là { hi (K) = (hi-1(K) +1) mod nB với 1   i   nB-1 và h0 là hàm băm cơ sở }.

Dạng cấu trúc băm sử dụng dây chuyền bucket được gọi là băm mở. Dạng sử dụng dãy các hàm băm được gọi là băm đóng. Trong các hệ CSDL, cấu trúc băm đóng thường được ưa dùng hơn.

Chỉ mục băm

Một chỉ mục băm tổ chức các khoá tìm kiếm cùng con trỏ kết hợp vào một cấu trúc file băm như sau: áp dụng một hàm băm trên khoá tìm kiếm để định danh bucket sau đó lưu giá trị khoá và con trỏ kết hợp vào bucket này (hoặc vào các bucket tràn). Chỉ mục băm thường là chỉ mục thứ cấp.

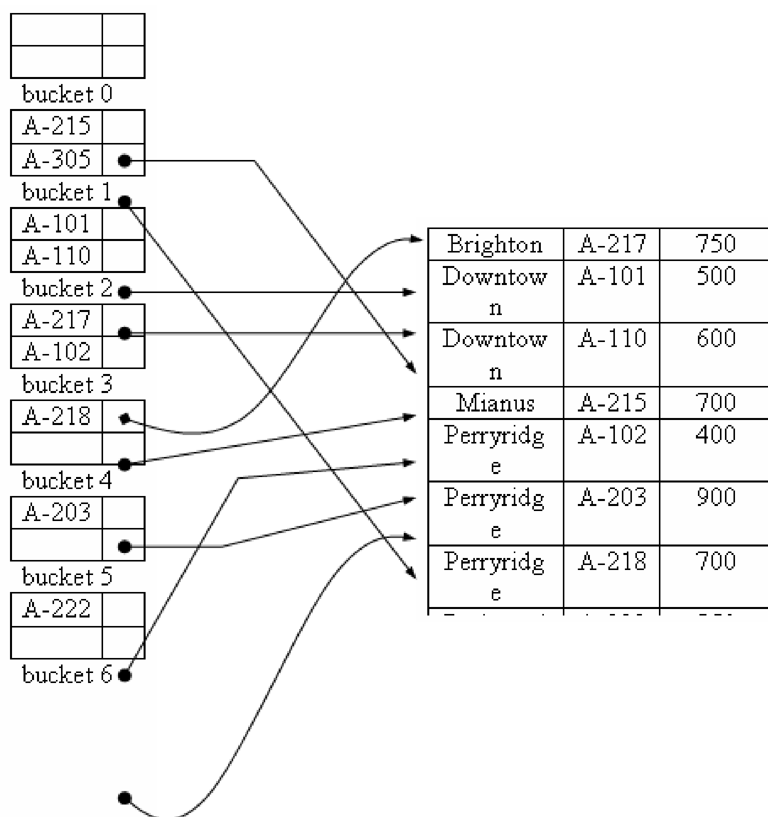
Hàm băm trên số tài khoản được tính theo công thức:

h(Account_number) = (tổng các chữ số trong số tài khoản) mod 7

BĂM ĐỘNG (Dynamic Hashing)

Trong kỹ thuật băm tĩnh (static hashing), tập B các địa chỉ bucket phải là cố định. Các CSDL phát triển lớn lên theo thời gian. Nếu ta sử dụng băm tĩnh cho CSDL, ta có ba lớp lựa chọn:

1. Chọn một hàm băm dựa trên kích cỡ file hiện hành. Sự lụa chọn này sẽ dẫn đến sự suy giảm hiệu năng khi CSDL lớn lên.
2. Chọn một hàm băm dựa trên kích cỡ file dự đoán trước cho một thời điểm nào đó trong tương lai. Mặc dù sự suy giảm hiệu năng được cải thiện, một lượng đáng kể không gian có thể bị lãng phí lúc khởi đầu.
3. Tổ chức lại theo chu kỳ cấu trúc băm đáp ứng sự phát triển kích cỡ file. Một sự tổ chức lại như vậy kéo theo việc lựa chọn một hàm băm mới, tính lại hàm băm trên mỗi mẩu tin trong file và sinh ra các gán bucket mới. Tổ chức lại là một hoạt động tốn thời gian. Hơn nũa, nó đòi hỏi cấm truy xuất file trong khi đang tổ chức lại file.

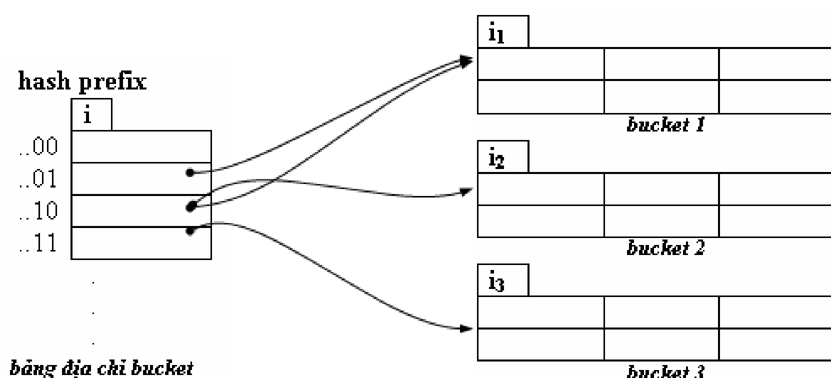


Chỉ mục băm trên khoá tìm kiếm account-number của file account

Kỹ thuật băm động cho phép sửa đổi hàm băm để phù hợp với sự tăng hoặc giảm của CSDL. Một dạng băm động được gọi là băm có thể mở rộng (extendable hashing) được thực hiện như sau: Chọn một hàm băm h với các tính chất đều, ngẫu nhiên và có miền giá trị tương đối rộng, chẳng hạn, là một số nguyên b bit (b thường là 32). Khi khởi đầu ta

không sử dụng toàn bộ b bit giá trị băm. Tại một thời điểm, ta chỉ sử dụng i bit 0   i   b. i bit này được dùng như một độ dời (offset) trong một bảng địa chỉ bucket phụ. giá trị i tăng lên hay giảm xuống tuỳ theo kích cỡ CSDL.

Số i xuất hiện bên trên bảng địa chỉ bucket chỉ ra rằng i bit của giá trị băm h(K) được đòi hỏi để xác định bucket đúng cho K, số này sẽ thay đổi khi kích cỡ file thay đổi. Mặc dù i bit dược đòi hỏi để tìm đầu vào đúng trong bảng địa chỉ bucket, một số đầu vào bảng kề nhau có thể trỏ đến cùng một bucket. Tất cả các như vậy có chung hash prefix chung, nhưng chiều dài của prefix này có thể nhỏ hơn i. Ta kết hợp một số nguyên chỉ độ dài của hash prefix chung này, ta sẽ ký hiệu số nguyên kết hợp với bucket j là ij. Số các đầu vào bảng địa chỉ bucket trỏ đến bucket j là 2^{i-i_j} .



Cấu trúc băm có thể mở rộng tổng quát

Để định vị bucket chứa giá trị khoá tìm kiếm K , ta lấy i bit cao đầu tiên của h(K), tìm trong đầu vào bảng tương ứng với chuỗi bit này và lần theo con trỏ trong đầu vào bảng này. Để xen một mẩu tin với giá trị khoá tìm kiếm K, tiến hành thủ tục dịnh vị trên, ta được bucket, giả sử là bucket j. Nếu còn cho cho mẩu tin, xen mẩu tin vào trong bucket đó. Nếu không, ta phải tách bucket ra và phân phối lại các mẩu tin hiện có cùng mẩu tin mới. Để tách bucket, đầu tiên ta xác định từ giá trị băm có cần tăng số bit lên hay không.

- Nếu i = ij , chỉ có một đầu vào trong bảng địa chỉ bucket trỏ đến bucket j. ta cần tăng kích cỡ của bảng địa chỉ bucket sao cho ta có thể bao hàm các con trỏ đến hai bucket kết quả của việc tách bucket j bằng cách xét thêm một bit của giá trị băm. tăng giá trị i lên một, như vậy kích cỡ của bảng địa chỉ bucket tăng lên gấp đôi. Mỗi một đầu vào được thay bởi hai đầu vào, cả hai cùng chứa con trỏ của đầu vào gốc. Bây giờ hai đầu vào trong bảng địa chỉ bucket trỏ tới bucket j. Ta định vị một bucket mới (bucket z), và đặt đầu vào thứ hai trỏ tới bucket mới, đặt ij và iz về i, tiếp theo đó mỗi một mẩu tin trong bucket j được băm lại, tuỳ thuộc vào i bit đầu tiên, sẽ hoặc ở lại bucket j hoặc được cấp phát cho bucket mới được tạo.

- Nếu i > ij khi đó nhiều hơn một đầu vào trong bảng địa chỉ bucket trỏ tới bucket j. như vậy ta có thể tách bucket j mà không cần tăng kích cỡ bảng địa chỉ bucket. Ta cấp phát một bucket mới (bucket z) và đặt ij và iz đến giá trị là kết quả của việc thêm 1 vào giá trị ij gốc. Kế đến, ta điều chỉnh các đầu vào trong bảng địa chỉ bucket trước đây trỏ tới bucket j. Ta để lại nửa đầu các đầu vào, và đặt tất cả các đầu vào còn lại trỏ tới bucket mới tạo (z). Tiếp theo, mỗi mẩu tin trong trong bucket j được băm lại và được cấp phát cho hoặc vào bucket j hoặc bucket z.

Để xoá một mẩu tin với giá trị khoá K, trước tiên ta thực hiện thủ tục định vị, ta tìm được bucket tương ứng, gọi là j, ta xoá cả khoá tìm kiếm trong bucket lẫn mẩu tin mẩu tin trong file. bucket cũng bị xoá, nếu nó trở nên rỗng. Chú ý rằng, tại điểm này, một số bucket có thể được kết hợp lại và kích cỡ của bảng địa chỉ bucket sẽ giảm đi một nửa.

Ưu điểm chính của băm có thể mở rộng là hiệu năng không bị suy giảm khi file tăng kích cỡ, hơn nữa, tổng phí không gian là tối tiểu mặc dù phải thêm vào không gian cho bảng địa chỉ bucket. Một khuyết điểm của băm có thể mở rộng là việc tìm kiếm phải bao hàm một mức gián tiếp: ta phải truy xuất bảng địa chỉ bucket trước khi truy xuất đến bucket. Vì vậy, băm có thể mở rộng là một kỹ thuật rất hấp dẫn.

CHỌN CHỈ MỤC HAY BĂM ?

Ta đã xét qua các sơ đồ: chỉ mục thứ tự, băm. Ta có thể tổ chức file các mẩu tin bởi hoặc sử dụng tổ chức file tuần tự chỉ mục, hoặc sử dụng B+-cây, hoặc sử dụng băm ... Mỗi sơ đồ có những các ưu điểm trong các tình huống nhất định. Một nhà thực thi hệ CSDL có thể cung cấp nhiều nhiều sơ đồ, để lại việc quyết định sử dụng sơ đồ nào cho nhà thiết kế CSDL. Để có một sự lựa chọn khôn ngoan, nhà thực thi hoặc nhà thiết kế CSDL phải xét các yếu tố sau:

- Cái giá phải trả cho việc tổ chức lại theo định kỳ của chỉ mục hoặc băm có chấp nhận được hay không?
- Tần số tương đối của các hoạt động xen và xoá là bao nhiêu ?
- Có nên tối ưu hoá thời gian truy xuất trung bình trong khi thời gian truy xuất trường hợp xấu nhất tăng lên hay không ?
- Các kiểu vấn tin mà các người sử dụng thích đặt ra là gì ?

CẤU TRÚC LƯU TRỮ CHO CSDL HƯỚNG ĐỐI TƯỢNG

SẮP XẾP CÁC ĐỐI TƯỢNG VÀO FILE

Phần dữ liệu của đối tượng có thể được lưu trữ bởi sử dụng các cấu trúc file được mô tả trước đây với một số thay đổi do đối tượng có kích cỡ không đều, hơn nữa đối tượng có thể rất lớn. Ta có thể thực thi các trường tập hợp ít phần tử bằng cách sử dụng danh sách liên kết, các trường tập hợp nhiều phần tử bởi B-cây hoặc bởi các quan hệ riêng biệt trong cơ sở dữ liệu. Các trường tập hợp cũng có thể bị loại trừ ở mức lưu trữ bởi chuẩn hoá. Các đối tượng cực lớn khó có thể phân tích thành các thành phần nhỏ hơn có thể được lưu trữ trong một file riêng cho mỗi đối tượng.

THỰC THI ĐỊNH DANH ĐỐI TƯỢNG

Vì đối tượng được nhận biết bởi định danh của đối tượng (OID = objject Identifier), Một hệ lưu trữ đối tượng cần phải có một cơ chế để tìm kiếm một đối tượng được cho bởi một OID. Nếu các OID là logic, có nghĩa là chúng không xác định vị trí của dối tượng, hệ thống lưu trữ phải duy trì một chỉ mục mà nó ánh xạ OID tới vị trí hiện hành của đối tượng. Nếu các OID là vật lý, có nghĩa là chúng mã hoá vị trí của đối tượng, đối tượng có thể dược tìm trực tiếp. Các OID điển hình có ba trường sau:

1. Một volume hoặc định danh file
2. Một định danh trang bên trong volume hoặc file
3. Một offset bên trong trang

Hơn nữa, OID vật lý có thẻ chứa một định danh duy nhất, nó là một số nguyên tách biệt OID với các định danh của các đối tượng khác đã được lưu trữ ở cùng vị trí trước đây và đã bị xoá hoặc dời đi. Định danh duy nhất này cũng được lưu với đối tượng, các định danh trong một OID và đối tượng tương ứng phù hợp. Nếu định danh duy nhất trong một OID vật lý không khớp với với định danh duy nhất trong đối tượng mà OID này trỏ tới, hệ thống phát hiện ra rằng con trỏ là bám và báo một lỗi. Lỗi con trỏ như vậy xảy ra khi OID vật lý tương ứng với đối tượng cũ đã bị xoá do tai nạn. Nếu không gian bị chiếm bởi đối tượng được cấp phát lại, có thể có một đối tượng mới ở vào vị trí này và có thể được định địa chỉ không đúng bởi định danh của đối tượng cũ. Nếu không phát hiện được, sử dụng con trỏ bám có thể gây nên sự sai lạc của một đối tượng mới được lưu ở cùng vị trí. Định danh duy nhất trợ giúp phát hiện lỗi như vậy. Giả sử một đối tượng phải di chuyển sang trang mới do sự lớn lên của đối tượng và trang cũ không có không gian phụ. Khi đó OID vật lý trỏ tới trang cũ bây giờ không còn chứa đối tượng. Thay vì thay đổi OID của đối tượng (điều này kéo theo sự thay đổi mỗi đối tượng trỏ tới đối tượng này) ta để địa chỉ forward ở vị trí cũ. Khi CSDL tìm đối tượng, nó tìm địa chỉ forward thay cho tìm đối tượng và sử dụng địa chỉ forward để tìm đối tượng.

QUẢN TRỊ CÁC CON TRỎ BỀN (persistent pointers)

Ta thực thi các con trỏ bền trong ngôn ngữ lập trình bền (persistent programming language) bằng cách sử dụng các OID. Các con trỏ bền có thể là các OID vật lý hoặc logic. Sự khác

nhau quan trọng giữa con trỏ bền và con trỏ trong bộ nhớ là kích thước concủa con trỏ. Con trỏ trong bộ nhớ chỉ cần đủ lớn để định địa chỉ toàn bộ bộ nhớ ảo, hiện tại kích cỡ con trỏ trong bộ nhớ là 4 byte. Con trỏ bền để định địa chỉ toàn bộ dữ liệu trong một CSDL, nên kích cỡ của nó ít nhất là 8 byte.

Pointer Swizzling

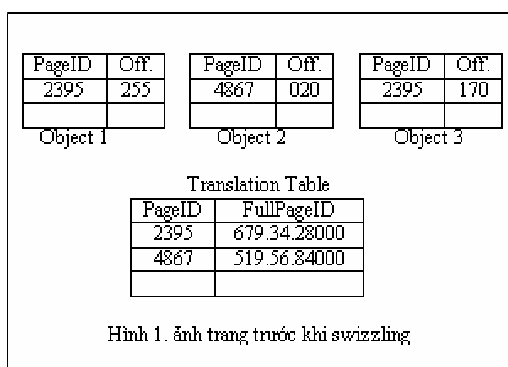
Hành động tìm một đối tượng được cho bởi định danh được gọi là dereferencing. Đã cho một con trỏ trong bộ nhớ, tìm đối tượng đơn thuần là một sự tham khảo bộ nhớ. Đã cho một con trỏ bền, dereferencing một đối tượng có một bước phụ: phải tìm vị trí hiện hành của đối tượng trong bộ nhớ bởi tìm con trỏ bền trong một bảng. Nếu đối tượng chưa nằm trong bộ nhớ, nó phải được nạp từ đĩa. Ta có thể thực thi bảng tìm kiếm này hoàn toàn hiệu quả bởi sử dụng băm, song tìm kiếm vẫn chậm.

pointer swizling là một phương pháp để giảm cái giá tìm kiếm các đối tượng bền đã hiện diện trong bộ nhớ. ý tưởng là khi một con trỏ bền được dereference, đối tượng được định vị và mang vào trong bộ (nhớ nếu nó chưa có ở đó). Bây giờ một bước phụ được thực hiện: một con trỏ trong bộ nhớ tới đối tượng được lưu vào vị trí của con trỏ bền. Lần kế con trỏ bền tương tự được dereference, vị trí trong bộ nhớ có thể được đọc ra trực tiếp. Trong trường hợp các đối tượng bền phải di chuyển lên đĩa để lấy không gian cho đối tượng bền khác, cần một bước phụ để đảm bảo đối tượng vẫn trong bộ nhớ cũng phải được thực hiện. Khi một đối tượng được viết ra. bất kỳ con trỏ bền nào mà nó chứa và bị swizzling phải được unswizzling như vậy được chuyển đổi về biểu diễn bền của chúng. pointer swizzling trên poiter dereferenc được mô tả này được gọi là software swizzling. Quan trị buffer sẽ phức tạp hơn nếu pointer swizzling được sử dụng.

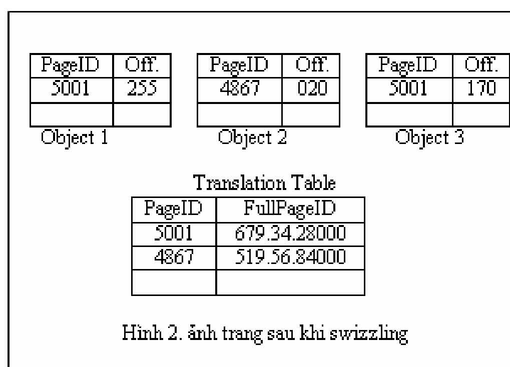
Hardware swizzling

Việc có hai kiểu con trỏ, con trỏ bền (persistent pointer) và con trỏ tạm (transient pointer / con trỏ trong bộ nhớ), là điều khá bất lợi. Người lập trình phải nhớ kiểu con trỏ và có thể phải viết mã chương trình hai lần- một cho các con trỏ bền và một cho con trỏ tạm. Sẽ thuận tiên hơn nếu cả hai kiểu con trỏ này cùng kiểu. Một cách đơn giản để trộn lẫn hai con rỏ này là mở rộng chiều dài con trỏ bộ nhớ cho bằng kích cỡ con trỏ bền và sử dụng một bit của phần định danh để phân biệt chúng. Cách làm này sẽ làm tăng chi phí lưu trữ đối với các con trỏ tạm. Ta sẽ mô tả một kỹ thuật được gọi là hardware swizzling nó sử dụng phần cứng quản trị bộ nhớ để giải quyết vấn đề này. Hardware swizzling có hai điểm lợi hơn so với software swizzling: Thứ nhất, nó cho phép lưu trữ các con trỏ bền trong đối tượng trong lượng không gian bằng với lượng không gian con trỏ bộ nhớ đòi hỏi. Thứ hai, nó chuyển đổi trong suốt giữa các con trỏ bền và các con trỏ tạm một cách thông minh và hiệu quả. Phần mềm được viết để giải quyết các con trỏ trong bộ nhớ có thể giải quyết các con trỏ bền mà không cần thay đổi.

hardware swizzling sử dụng sự biểu diễn các con trỏ bền được chứa trong đối tượng trên đĩa như sau: Một con trỏ bền được tách ra thành hai phần, một là định danh trang và một là offset bên trong trang. Định danh trang thường là một con trỏ trực tiếp nhỏ: mỗi trưng có một bảng dịch (translation table) cung cấp một ánh xạ từ các định danh trang ngắn đến các định danh CSDL đầy đủ. Hệ thống phải tìm định danh trang nhỏ trong một con trỏ bền trong bảng dịch để tìm định danh trang đầy đủ. Bảng dịch, trong trường hợp xấu nhất, chỉ lớn bằng số tối đa các con trỏ có thể được chứa trong các đối tượng trong một trang. Với một trang kích thước 4096 byte, con trỏ kích thước 4 byte, số tối đa các con trỏ là 1024. Trong thực tế số tối đa nhỏ hơn con số này rất nhiều. Định danh trang nhỏ chỉ cần đủ số bit để định danh một dòng trong bảng, nếu số dòng tối đa là 1024, chỉ cần 10 bit để định danh trang nhỏ. Bảng dịch cho phép toàn bộ một con trỏ bền lấp đầy một không gian bằng không gian cho một con trỏ trong bộ nhớ.



Hình 1. ảnh trang trước khi swizzling



Hình 2. ảnh trang sau khi swizzling

Trong hình 1, trình bày sơ đồ biểu diễn con trỏ bền, có ba đối tượng trong trang, mỗi một chứa một con trỏ bền. Bảng dịch cho ra ánh xạ giữa định danh trang ngắn và định danh trang CSDL đầy đủ đối với mỗi định danh trang ngắn trong các con trỏ bền này. Định danh trang CSDL được trình bày dưới dạng volume.file.offset. Thông tin phụ được duy trì với mỗi trang sao cho tất cả các con trỏ bền trong trang có thể tìm thấy. Thông tin được cập nhất khi một đối tượng được tạo ra hay bị xoá khỏi trang. Khi một con trỏ trong bộ nhớ được dereferencing, nếu hệ điều hành phát hiện trang trong không gian địa chỉ ảo được trỏ tới không được cấp phát lưu trữ hoặc có truy xuất được bảo vệ, khi đó một sự vi phạm đoạn được ước đoán là xảy ra. Nhiều hệ điều hành cung cấp một cơ chế xác định một hàm sưe được gọi khi vi phạm đoạn xảy ra, một cơ chế cấp phát lưu trữ cho các trang trong không gian địa chỉ ảo, và một tập các quyền truy xuất trang. Đầu tiên, ta xét một con trỏ trong bộ nhớ trỏ tới trang v được khử tham chiếu, khi lưu trữ chưa được cấp phát cho trang này. Một vi phạm đoạn sẽ xảy ra và kết quả là một lời gọi hàm trên hệ CSDL. Hệ CSDL dầu tiên xác định trang CSDL nào đã được cấp phát cho trang bộ nhớ ảo v, gọi định danh trang đầy đủ của trang CSDL là P, nếu không có trang CSDL cấp phát cho v, một lỗi được thông báo., nếu không, hệ CSDL cấp phát không gian lưu trữ cho trang v và nạp trang CSDL P vào trong v. Pointer swizzling bây giờ được làm đối với trang P như sau: Hệ thông tìm tất cả các con trỏ bèn được chứa trong các đối tương trong trang, bằng cách sử dụng thông tin phụ được lưu trữ trong trang. Ta xét một con trỏ như vậy và gọi nó là (pi, oi), trong đó pi là định danh trang ngắn và oi là offset trong trang. Giả sử Pi là định danh trang đầy đủ của pi được tìm thấy trong bảng dịch trong trang P. Nếu trang Pi chưa có một trang

bộ nhớ ảo được cấp cho nó, một trang tự do trong không gian địa chỉ ảo sẽ được cấp cho nó. Trang Pi sẽ nằm ở vị trí địa chỉ ảo nàynếu và khi nó được mang vào. Tại điểm này, trang trong không gian địa chỉ ảo không có bất kỳ một lưu trữ nào được cấp cho nó, cả trong bộ nhớ lẫn trên đĩa, đơn tuần chỉ là một khoảng địa chỉ dự trữ cho trang CSDL. Bây giờ giả sử trang bộ nhớ ảo đã được cấp phát cho Pi là vi . Ta cập nhật con trỏ (pi, oi) bởi thay thế pi bởi vi , cuối cùng sau khi swizzling tất cả các con trỏ bền trong P, sự khử tham chiếu gây ra vi phạm đoạn được cho phép tiếp tục và sẽ tìm thấy đối tượng đang được tìm kiếm trong bộ nhớ.

Trong hình 2, trình bày trạng thái trang trong hình 1 sau khi trang này được mang vào trong bộ nhớ và các con trỏ trong nó đã được swizzling. ở đây ta giả thiết trang định danh trang CSDL của nó là 679.34.28000 được ánh xạ đến trang 5001 trong bộ nhớ, trong khi trang định danh của nó là 519.56.84000 được ánh xạ dến trang 4867. Tất cả các con trỏ trong đối tượng đã được cập nhật để phản ánh tương ứng mới và bây giờ có thể được dùng như con trỏ trong bộ nhớ. ở cuối của giai đoạn dịch đói với một trang, các đối tượng trong trang thoả mãn một tính chất quan trọng: Tất cả các con trỏ bền được chứa trong đối tượng trong trang được chuyển đổi thành các con trỏ trong bộ nhớ.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG III

III.1Xét sự sắp xếp các khối dữ liệu và các khối parity trên bốn đĩa sau:

Đĩa 1	Đĩa 2	Đĩa 3	Đĩa 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
⋮	⋮	⋮	⋮

Trong đó các Bi biểu diễn các khối dữ liệu, các khối Pi biểu diễn các khối parity. Khối Pi là khối parity đối với các khối dữ liệu B4i - 3 , B4i - 2 , B4i - 1 , B4i . Hãy nêu các vấn đề gặp phải của cách sắp xếp này.

III.2Một sự mất điện xảy ra trong khi một khối đang được viết sẽ dẫn tới kết quả là khối đó có thể chỉ được viết một phần. Giả sử rằng khối được viết một phần có thể phát hiện được. Một viết khối nguyên tử là hoặc toàn bộ khối được viết hoặc không có gì được viết (không có khối được viết một phần). Hãy đề nghị những sơ đồ để có được các viết khối nguyên tử hiệu quả trên các sơ đồ RAID:

1. Mức 1 (mirroring)
2. Mức 5(block interleaved, distributed parity)

III.3Các hệ thống RAID tiêu biểu cho phép thay thế các đĩa hư không cần ngưng truy xuất hệ thống. Như vậy dữ liệu trong đĩa bị hư sẽ phải được tái tạo và viết lên đĩa thay thế trong khi hệ thống vẫn tiếp tục hoạt động. Với mức RAID nào thời lượng giao thoa giữa việc tái tạo và các truy xuất đĩa còn đang chạy là ít nhất ? Giải thích.

III.4Xét việc xoá mẩu tin 5 trong file:

0	**Perryridge**	A-102	400
1	Round Hill	A-305	350
8	Perryridge	A-218	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600

So sánh các điều hay/dở tương đối của các kỹ thuật xoá sau:

1. Di chuyển mẩu tin 6 đến không gian chị chiếm bởi mẩu tin 5, rồi di chuyển mẩu tin 7 đến chỗ bị chiếm bởi mẩu tin 6.
2. Di chuyển mẩu tin 7 đến chỗ bị chiếm bởi mẩu tin 5
3. Đánh dấu xoá mẩu tin 5.

III.5Vẽ cấu trúc của file:

header				
0	Perryridge	A-102	400	
1				
2	Mianus	A-215	700	
3	Downtown	A-101	500	
4				
5	Perryridge	A-201	900	
6				
7	Downtown	A-110	600	
8	Perryridge	A-218	700	

Sau mỗi bước sau:

1. Insert(Brighton, A-323, 1600)
2. Xoá mẩu tin 2
3. Insert(Brighton, A-636, 2500)

III.6Vẽ lại cấu trúc file:

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥				
2	Mianus	A-101	800	⊥				
3	Downtown	A-211	500	A-222	600	⊥		
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥				

Sau mỗi bước sau:

1. Insert(Mianus, A-101, 2800)
2. Insert(Brighton, A-323, 1600)
3. Delete (Perryridge, A-102, 400)

III.7Điều gì sẽ xảy ra nếu xen mẩu tin (Perryridge, A-999, 5000) vào file trong III.6.

III.8Vẽ lại cấu trúc file dưới đây sau mỗi bước sau:

1. Insert(Mianus, A-101, 2800
2. Insert(Brighton, A-323, 1600)
3. Delete (Perryridge, A-102, 400)

0	Perryridge	A-102	400	•
1		A-201	900	•
2		A-210	700	•
3	Round Hill	A-301	350	•
4	Mianus	A-101	800	•
5	Downtown	A-211	500	
6	Redwood	A-300	650	•
7	Brighton	A-111	750	•

III.9Nêu lên một ví dụ, trong đó phương pháp không gian dự trữ để biểu diễn các mẩu tin độ dài thay đổi phù hợp hơn phương pháp con trỏ.

III.10Nêu lên một ví dụ, trong đó phương pháp con trỏ để biểu diễn các mẩu tin độ dài thay đổi phù hợp hơn phương pháp không gian dự trữ.

III.11Nếu một khối trở nên rỗng sau khi xoá. Khối này được tái sử dụng vào mục đích gì ?

III.12Trong tổ chức file tuần tự, tại sao khối tràn được sử dụng thậm chí, tại thời điểm đang xét, chỉ có một mẩu tin tràn ?

III.13 Liệt kê các ưu điểm và nhược điểm của mỗi một trong các chiến lược lưu trữ CSDL quan hệ sau:

1. Lưu trữ mỗi quan hệ trong một file
2. Lưu trữ nhiều quan hệ trong một file

III.14Nêu một ví dụ biểu thức đại số quan hệ và một chiến lược xử lý vấn tin trong đó:

1. MRU phù hợp hơn LRU
2. LRU phù hợp hơn MRU

III.15Khi nào sử dụng chỉ mục đặc phù hợp hơn chỉ mục thưa ? Giải thích.

III.16Nêu các điểm khác nhau giữa chỉ mục sơ cấp và chỉ mục thứ cấp .

III.17Có thể có hai chỉ mục sơ cấp đối với hai khoá khác nhau trên cùng một quan hệ ? Giải thích.

III.18Xây dựng một B+-cây đối với tập các giá trị khoá: (2, 3, 5, 7, 11, 15, 19, 25, 29, 33, 37, 41, 47). Giả thiết ban đầu cây là rỗng và các giá trị được xen theo thứ tự tăng. Xét trong các trường hợp sau:

1. Mỗi nút chứa tối đa 4 con trỏ
2. Mỗi nút chứa tối đa 6 con trỏ
3. Mỗi nút chứa tối đa 8 con trỏ

III.19Đối với mỗi B+-cây trong bài tập III.18 Bày tỏ các bước thực hiện trong các vấn tin sau:

1. Tìm mẩu tin với giá trị khoá tìm kiếm 11
2. Tìm các mẩu tin với giá trị khoá nằm trong khoảng [7..19]

III.20Đối với mỗi B+-cây trong bài tập III.18. Vẽ cây sau mỗi một trong dãy hoạt động sau:

1. Insert 9
2. Insert 11
3. Insert 11
4. Delete 25

5. Delete 19

III.21Cùng câu hỏi như trong III.18 nhưng đối với B-cây

III.22 Nêu và giải thích sự khác nhau giữa băm đóng và băm mở. Nêu các ưu, nhược điểm của mỗi kỹ thuật này.

III.23Điều gì gây ra sự tràn bucket trong một tổ chức file băm ? Làm gì để giảm sự tràn này ?

III.24Giả sử ta đang sử dụng băm có thể mở rộng trên một trên một file chứa các mẩu tin với các giá trị khoá tìm kiếm sau:

2, 3, 5, 7, 11, 17, 19, 23, 37, 31, 35, 41, 49, 55

Vẽ cấu trúc băm có thể mở rộng đối với file này nếu hàm băm là h(x) = x mod 8 và mỗi bucket có thể chứa nhiều nhất được ba mẩu tin.

III.25Vẽ lại cấu trúc băm có thể mở rộng trong bài tập III.24 sau mỗi bước sau:

1. Xoá 11
2. Xoá 55
3. Xen 1
4. Xen 15

Giao dịch
MỤC ĐÍCH Giới thiệu khái niệm giao dịch, các tính chất một giao dịch cần phải có để sự hoạt động của nó trong môi trường có "biến động" vẫn đảm bảo cho CSDL luôn ở trạng thái nhất quán. Giới thiệu các khái niệm khả tuần tự, khả tuần tự xung đột, khả tuần tự view, khả phục hồi và cascadeless, các thuật toán kiểm thử tính khả tuần tự xung đột và khả tuần tự view YÊU CẦU Hiểu khái niệm giao dịch, các tính chất cần phải có của giao dịch và "ai" là người đảm bảo các tính chất đó Hiểu các khái niệm về khả tuần tự, khả phục hồi và mối tương quan giữa chúng Hiểu và vận dụng các thuật toán kiểm thử

KHÁI NIỆM

Một giao dịch là một đơn vị thực hiện chương trình truy xuất và có thể cập nhật nhiều hạng mục dữ liệu. Một giao dịch thường là kết quả của sự thực hiện một chương trình người dùng được viết trong một ngôn ngữ thao tác dữ liệu mức cao hoặc một ngôn ngữ lập trình (SQL, COBOL, PASCAL ...), và được phân cách bởi các câu lệnh (hoặc các lời gọi hàm) có dạng begin transaction và end transaction. Giao dịch bao gồm tất cả các hoạt động được thực hiện giữa begin và end transaction.

Để đảm bảo tính toàn vẹn của dữ liệu, ta yêu cầu hệ CSDL duy trì các tính chất sau của giao dịch:

- Tính nguyên tử (Atomicity). Hoặc toàn bộ các hoạt động của giao dịch được phản ánh đúng đắn trong CSDL hoặc không có gì cả.
- Tính nhất quán (consistency). Sự thực hiện của một giao dịch là cô lập (Không có giao dịch khác thực hiện đồng thời) để bảo tồn tính nhất quán của CSDL.
- Tính cô lập (Isolation). Cho dù nhiều giao dịch có thể thực hiện đồng thời, hệ thống phải đảm bảo rằng đối với mỗi cặp giao dịch Ti, Tj , hoặc Tj kết thúc thực hiện trước khi Ti khởi động hoặc Tj bắt đầu sự thực hiện sau khi Ti kết thúc. Như vậy mỗi giao dịch không cần biết đến các giao dịch khác đang thực hiện đồng thời trong hệ thống.

- Tính bền vững (Durability). Sau một giao dịch hoàn thành thành công, các thay đổi đã được tạo ra đối với CSDL vẫn còn ngay cả khi xảy ra sự cố hệ thống.

Các tính chất này thường được gọi là các tính chất ACID (Các chữ cái đầu của bốn tính chất). Ta xét một ví dụ: Một hệ thống nhà băng gồm một số tài khoản và một tập các giao dịch truy xuất và cập nhật các tài khoản. Tại thời điểm hiện tại, ta giả thiết rằng CSDL nằm trên đĩa, nhưng một vài phần của nó đang nằm tạm thời trong bộ nhớ. Các truy xuất CSDL được thực hiện bởi hai hoạt động sau:

- READ(X). chuyển hạng mục dữ liệu X từ CSDL đến buffer của giao dịch thực hiện hoạt động READ này.
- WRITE(X). chuyển hạng mục dữ liệu X từ buffer của giao dịch thực hiện WRITE đến CSDL.

Trong hệ CSDL thực, hoạt động WRITE không nhất thiết dẫn đến sự cập nhật trực tiếp dữ liệu trên đĩa; hoạt động WRITE có thể được lưu tạm thời trong bộ nhớ và được thực hiện trên đĩa muộn hơn. Trong ví dụ, ta giả thiết hoạt động WRITE cập nhật trực tiếp CSDL.

Ti là một giao dịch chuyển 50 từ tài khoản A sang tài khoản B. Giao dịch này có thể được xác định như sau:

Ti : READ(A);

A:=A - 50;

WRITE(A)

READ(B);

B:=B + 50;

WRITE(B);

figure IV-

Ta xem xét mỗi một trong các yêu cầu ACID

- Tính nhất quán: Đòi hỏi nhất quán ở đây là tổng của A và B là không thay đổi bởi sự thực hiện giao dịch. Nếu không có yêu cầu nhất quán, tiền có thể được tạo ra hay bị phá huỷ bởi giao dịch. Dễ dàng kiểm nghiệm rằng nếu CSDL nhất quán trước một thực hiện giao dịch, nó vẫn nhất quán sau khi thực hiện giao dịch. Đảm bảo tính nhất quán cho một giao dịch là trách nhiệm của người lập trình ứng dụng người đã viết ra giao dịch. Nhiệm vụ này có thể được làm cho dễ dàng bởi kiểm thử tự động các ràng buộc toàn vẹn.
- Tính nguyên tử: Giả sử rằng ngay trước khi thực hiện giao dịch Ti, giá trị của các tài khoản A và B tương ứng là 1000 và 2000. Giả sử rằng trong khi thực hiện giao dịch Ti, một sự cố xảy ra cản trở Ti hoàn tất thành công sự thực hiện của nó. Ta cũng giả sử rằng sự cố xảy ra sau khi hoạt động WRITE(A) đã được thực hiện, nhưng trước khi hoạt động WRITE(B) được thực hiện. Trong trường hợp này giá trị của tài khoản A và B là 950 và 2000. Ta đã phá huỷ 50$. Tổng A+B không còn được bảo tồn.

Như vậy, kết quả của sự cố là trạng thái của hệ thống không còn phản ánh trạng thái của thế giới mà CSDL được giả thiết nắm giữ. Ta sẽ gọi trạng thái như vậy là trạng thái không nhất quán. Ta phải đảm bảo rằng tính bất nhất này không xuất hiện trong một hệ CSDL. Chú ý rằng, cho dù thế nào tại một vài thời điểm, hệ thống cũng phải ở trong trạng thái không nhất quán. Ngay cả khi giao dịch Ti , trong quá trình thực hiện cũng tồn tại thời điểm tại đó giá trị của tài khoản A là 950 và tài khoản B là 2000 – một trạng thái không nhất quán. Trạng thái này được thay thế bởi trạng thái nhất quán khi giao dịch đã hoàn tất. Như vậy, nếu giao dịch không bao giờ khởi động hoặc được đảm bảo sẽ hoàn tất, trạng thái không nhất quán sẽ không bao giờ xảy ra. Đó chính là lý do có yêu cầu về tính nguyên tử: Nếu tính chất nguyên tử được cung cấp, tất cả các hành động của giao dịch được phản ánh trong CSDL hoặc không có gì cả. ý tưởng cơ sở để đảm bảo tính nguyên tử là như sau: hệ CSDL lưu vết (trên đĩa) các giá trị cũ của bất kỳ dữ liệu nào trên đó giao dịch đang thực hiện viết, nếu giao dịch không hoàn tất, giá trị cũ được khôi phục để đặt trạng thái của hệ thống trở lại trạng thái trước khi giao dịch diễn ra.

Đảm bảo tính nguyên tử là trách nhiệm của hệ CSDL, và được quản lý bởi một thành phần được gọi là thành phần quản trị giao dịch (transaction-management component).

- Tính bền vững: Tính chất bền vững đảm bảo rằng mỗi khi một giao dịch hoàn tất, tất cả các cập nhật đã thực hiện trên cơ sở dữ liệu vẫn còn đó, ngay cả khi xảy ra sự có hệ thống sau khi giao dịch đã hoàn tất. Ta giả sử một sự cố hệ thống có thể gây ra việc mất dữ liệu trong bộ nhớ chính, nhưng dữ liệu trên đĩa thì không mất. Có thể đảm bảo tính bền vững bởi việc đảm bảo hoặc các cập nhật được thực hiện bởi giao dịch đã được viết lên đĩa trước khi giao dịch kết thúc hoặc thông tin về sự cập nhật được thực hiện bởi giao dịch và được viết lên đĩa đủ cho phép CSDL xây dựng lại các cập nhật khi hệ CSDL được khởi động lại sau sự cố. Đảm bảo tính bền vững là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị phục hồi (recovery-management component). Hai thành phần quản trị giao dịch và quản trị phục hồi quan hệ mật thiết với nhau.
- Tính cô lập: Ngay cả khi tính nhất quán và tính nguyên tử được đảm bảo cho mỗi giao dịch, trạng thái không nhất quán vẫn có thể xảy ra nếu trong hệ thống có một số giao dịch được thực hiện đồng thời và các hoạt động của chúng đan xen theo một cách không mong muốn. Ví dụ, CSDL là không nhất quán tạm thời trong khi giao dịch chuyển khoản từ A sang B đang thực hiện, nếu một giao dịch khác thực hiện đồng thời đọc A và B tại thời điểm trung gian nàyvà tính A+B, nó đã tham khảo một giá trị không nhất quán, sau đó nó thực hiện cập nhật A và B dựa trên các giá trị không nhất quán này, như vậy CSDL có thể ở trạng thái không nhất quán ngay cả khi cả hai giao dịch hoàn tất thành công. Một giải pháp cho vấn đề các giao dịch thực hiện đồng thời là thực hiện tuần tự các giao dịch, tuy nhiên giải pháp này làm giả hiệu năng của hệ thống. Các giải pháp khác cho phép nhiều giao dịch thực hiện cạnh tranh đã được phát triển ta sẽ thảo luận về chúng sau này. Tính cô lập của một giao dịch đảm bảo rằng sự thực hiện đồng thời các giao dịch dẫn đến một trạng thái hệ thống tương đương với một trạng thái có thể nhận được bởi thực hiện các giao dịch này một tại một thời điểm theo một thứ nào đó.

Đảm bảo tính cô lập là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị cạnh tranh (concurrency-control component).

TRẠNG THÁI GIAO DỊCH

Nếu không có sự cố, tất cả các giao dịch đều hoàn tất thành công. Tuy nhiên, một giao dịch trong thực tế có thể không thể hoàn tất sự thực hiện của nó. Giao dịch như vậy được gọi là bị bỏ dở. Nếu ta đảm bảo được tính nguyên tử, một giao dịch bị bỏ dở không được phép làm ảnh hưởng tới trạng thái của CSDL. Như vậy, bất kỳ thay đổi nào mà giao dịch bị bỏ dở này phải bị huỷ bỏ. Mỗi khi các thay đổi do giao dịch bị bỏ dở bị huỷ bỏ, ta nói rằng giao dịch bị cuộn lại (rolled back). Việc này là trách nhiệm của sơ đồ khôi phục nhằm quản trị các giao dịch bị bỏ dở. Một giao dịch hoàn tất thành công sự thực hiện của nó được gọi là được bàn giao (committed). Một giao dịch được bàn giao (committed), thực hiện các cập nhật sẽ biến đổi CSDL sang một trạng thái nhất quán mới và nó là bền vững ngay cả khi có sự cố. Mỗi khi một giao dịch là được bàn giao (committed), ta không thể huỷ bỏ các hiệu quả của nó bằng các bỏ dở nó. Cách duy nhất để huỷ bỏ các hiệu quả của một giao dịch được bàn giao (committed) là thực hiện một giao dịch bù (compensating transaction); nhưng không phải luôn luôn có thể tạo ra một giao dịch bù. Do vậy trách nhiệm viết và thực hiện một giao dịch bù thuộc về người sử dụng và không được quản lý bởi hệ CSDL.

Một giao dịch phải ở trong một trong các trạng thái sau:

- Hoạt động (Active). Trạng thái khởi đầu; giao dịch giữ trong trạng thái này trong khi nó đang thực hiện.
- được bàn giao bộ phận (Partially Committed). Sau khi lệnh cuối cùng được thực hiện.
- Thất bại (Failed). Sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- Bỏ dở (Aborted). Sau khi giao dịch đã bị cuộn lại và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao dịch.
- được bàn giao (Committed). Sau khi hoàn thành thành công giao dịch.

Ta nói một giao dịch đã được bàn giao (committed) chỉ nếu nó đã di vào trạng thái Committed, tương tự, một giao dịch bị bỏ dở nếu nó đã đi vào trạng thái Aborted. Một giao dịch được gọi là kết thúc nếu nó hoặc là committed hoặc là Aborted. Một giao dịch khởi đầu bởi trạng thái Active. Khi nó kết thúc lệnh sau cùng của nó, nó chuyển sang trạng thái partially committed. Tại thời điểm này, giao dịch đã hoàn thành sự thực hiện của nó, nhưng nó vẫn có thể bị bỏ dở do đầu ra hiện tại vẫn có thể trú tạm thời trong bộ nhớ chính và như thế một sự cố phần cứng vẫn có thể ngăn cản sự hoàn tất của giao dịch. Hệ CSDL khi đó đã kịp viết lên đĩa đầy đủ thông tin giúp việc tái tạo các cập nhật đã được thực hiện trong quá trình thực hiện giao dịch, khi hệ thống tái khởi động sau sự cố. Sau hhi các thông tin sau cùng này được viết lên đĩa, giao dịch chuyển sang trạng thái committed.

Biểu đồ trạng thái tương ứng với một giao dịch như sau:

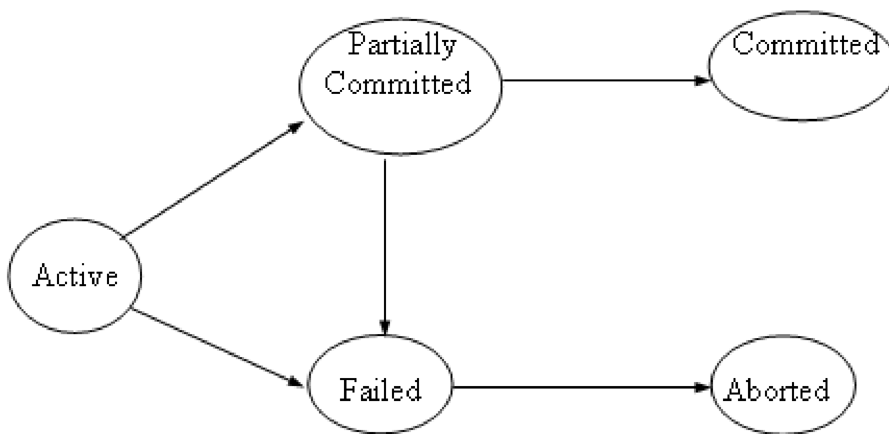


figure IV-

Với giả thiết sự cố hệ thống không gây ra sự mất dữ liệu trên đĩa, Một giao dịch đi vào trạng thái Failed sau khi hệ thống xác định rằng giao dịch không thể tiến triển bình thường được nữa (do lỗi phần cứng hoặc phần mềm). Như vậy, giao dịch phải được cuộn lại rồi chuyển sang trạng thái bỏ dở. Tại điểm này, hệ thống có hai lựa chọn:

- Khởi động lại giao dịch, nhưng chỉ nếu giao dịch bị bở dở là do lỗi phần cứng hoặc phần mềm nào đó không liên quan đến logic bên trong của giao dịch. Giao dịch được khởi động lại được xem là một giao dịch mới.
- Giết giao dịch thường được tiến hành hoặc do lỗi logic bên trong giao dịch, lỗi này cần được chỉnh sửa bởi viết lại chương trình ứng dụng hoặc do đầu vào xấu hoặc do dữ liệu mong muốn không tìm thấy trong CSDL.

Ta phải thận trọng khi thực hiện viết ngoài khả quan sát (observable external Write - như viết ra terminal hay máy in). Mỗi khi một viết như vậy xẩy ra, nó không thể bị xoá do nó có thể phải giao tiếp với bên ngoài hệ CSDL. Hầu hết các hệ thống cho phép các viết như thế xẩy ra chỉ khi giao dịch đã di vào trạng thái committed. Một cách để thực thi một sơ đồ như vậy là cho hệ CSDL lưu trữ tạm thời bất kỳ giá trị nào kết hợp với các viết ngoài như vậy trong lưu trữ không hay thay đổi và thực hiện các viết hiện tại chỉ sau khi giao dịch đã đi vào trạng thái committed. Nếu hệ thống thất bại sau khi giao dịch đi vào trạng thái committed nhưng trước khi hoàn tất các viết ngoài, hệ CSDL sẽ làm các viết ngoài này (sử dụng dữ liệu trong lưu trữ không hay thay đổi) khi hệ thống khởi động lại.

Trong một số ứng dụng, có thể muốn cho phép giao dịch hoạt động trình bày dữ liệu cho người sử dụng, đặc biệt là các giao dịch kéo dài trong vài phút hay vài giờ. Ta không thể cho phép xuất ra dữ liệu khả quan sát như vậy trừ phi ta buộc phải làm tổn hại tính nguyên tử giao dịch. Hầu hết các hệ thống giao dịch hiện hành đảm bảo tính nguyên tử và do vậy cấm dạng trao đổi với người dùng này.

THỰC THI TÍNH NGUYÊN TỬ VÀ TÍNH BỀN VỮNG

Thành phần quản trị phục hồi của một hệ CSDL hỗ trợ tính nguyên tử và tính bền vững. Trước tiên ta xét một sơ đồ đơn giản (song cực kỳ thiếu hiệu quả). Sơ đồ này giả thiết rằng chỉ một giao dịch là hoạt động tại một thời điểm và được dựa trên tạo bản sao của CSDL được gọi là các bản sao bóng (shadow copies). Sơ đồ giả thiết rằng CSDL chỉ là một file

trên đĩa. Một con trỏ được gọi là db_pointer được duy trì trên đĩa; nó trỏ tới bản sao hiện hành của CSDL.

Trong sơ đồ CSDL bóng (shadow-database), một giao dịch muốn cập nhật CSDL, đầu tiên tạo ra một bản sao đầy đủ của CSDL. Tất cả các cập nhật được làm trên bản sao này, không đụng chạm tới bản gốc (bản sao bóng). Nếu tại một thời điểm bất kỳ giao dịch bị bỏ dở, bản sao mới bị xoá. Bản sao cũ của CSDL không bị ảnh hưởng. Nếu giao dịch hoàn tất, nó được được bàn giao (committed) như sau. Đầu tiên, Hỏi hệ điều hành để đảm bảo rằng tất cả các trang của bản sao mới đã được viết lên đĩa (flush). Sau khi flush con trỏ db_pointer được cập nhật để trỏ đến bản sao mới; bản sao mới trở thành bản sao hiện hành của CSDL. Bản sao cũ bị xoá đi. Giao dịch được gọi là đã được được bàn giao (committed) tại thời điểm sự cập nhật con trỏ db_pointer được ghi lên đĩa. Ta xét kỹ thuật này quản lý sự cố giao dịch và sự cố hệ thống ra sao? Trước tiên, ta xét sự cố giao dịch. Nếu giao dịch thất bại tại thời điểm bất kỳ trước khi con trỏ db_pointer được cập nhật, nội dung cũ của CSDL không bị ảnh hưởng. Ta có thể bỏ dở giao dịch bởi xoá bản sao mới. Mỗi khi giao dịch được được bàn giao (committed), tất cả các cập nhật mà nó đã thực hiện là ở trong CSDL được trỏ bởi db_pointer. Như vậy, hoặc tất cả các cập nhật của giao dịch đã được phản ánh hoặc không hiệu quả nào được phản ánh, bất chấp tới sự cố giao dịch. Bây giờ ta xét sự cố hệ thống. Giả sử sự cố hệ thống xảy ra tại thời điểm bất kỳ trước khi db_pointer đã được cập nhật được viết lên đĩa. Khi đó, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và như vậy sẽ thấy nội dung gốc của CSDL – không hiệu quả nào của giao dịch được nhìn thấy trên CSDL. Bây giờ lại giả sử rằng sự cố hệ thống xảy ra sau khi db_pointer đã được cập nhật lên đĩa. Trước khi con trỏ được cập nhật, tất cả các trang được cập nhật của bản sao mới đã được viết lên đĩa. Từ giả thiết file trên đĩa không bị hư hại do sự cố hệ thống. Do vậy, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và sẽ thấy nội dung của CSDL sau tất cả các cập nhật đã thực hiện bởi giao dịch. Sự thực thi này phụ thuộc vào việc viết lên db_pointer, việc viết này phải là nguyên tử, có nghĩa là hoặc tất cả các byte của nó được viết hoặc không byte nào được viết. Nếu chỉ một số byte của con trỏ được cập nhật bởi việc viết nhưng các byte khác thì không thì con trỏ trở thành vô nghĩa và cả bản cũ lẫn bản mới của CSDL

có thể tìm thấy khi hệ thống khởi động lại. May mắn thay, hệ thống đĩa cung cấp các cập nhật nguyên tử toàn bộ khối đĩa hoặc ít nhất là một sector đĩa. Như vậy hệ thống đĩa đảm bảo việc cập nhật con trỏ db_pointer là nguyên tử. Tính nguyên tử và tính bền vững của giao dịch được đảm bảo bởi việc thực thi bản sao bóng của thành phần quản trị phục hồi. Sự thực thi này cực kỳ thiếu hiệu quả trong ngữ cảnh CSDL lớn, do sự thực hiện một giao dịch đòi hỏi phải sao toàn bộ CSDL. Hơn nữa sự thực thi này không cho phép các giao dịch thực hiện đồng thời với các giao dịch khác. Phương pháp thực thi tính nguyên tử và tính lâu bền mạnh hơn và đỡ tốn kém hơn được trình bày trong chương hệ thống phục hồi.

CÁC THỰC HIỆN CẠNH TRANH

Hệ thống xử lý giao dịch thường cho phép nhiều giao dịch thực hiện đồng thời. Việc cho phép nhiều giao dịch cập nhật dữ liệu đồng thời gây ra những khó khăn trong việc bảo đảm sự nhất quán dữ liệu. Bảo đảm sự nhất quán dữ liệu mà không đếm xỉa tới sự thực hiện cạnh tranh các giao dịch sẽ cần thêm các công việc phụ. Một phương pháp dễ tiến hành là cho các giao dịch thực hiện tuần tự: đảm bảo rằng một giao dịch khởi động chỉ sau khi giao dịch trước đã hoàn tất. Tuy nhiên có hai lý do hợp lý để thực hiện cạnh tranh là:

- Một giao dịch gồm nhiều bước. Một vài bước liên quan tới hoạt động I/O; các bước khác liên quan đến hoạt động CPU. CPU và các đĩa trong một hệ thống có thể hoạt động song song. Do vậy hoạt động I/O có thể được tiến hành song song với xử lý tại CPU. Sự song song của hệ thống CPU và I/O có thể được khai thác để chạy nhiều giao dịch song song. Trong khi một giao dịch tiến hành một hoạt động đọc/viết trên một đĩa, một giao dịch khác có thể đang chạy trong CPU, một giao dịch thứ ba có thể thực hiện đọc/viết trên một đĩa khác ... như vậy sẽ tăng lượng đầu vào hệ thống có nghĩa là tăng số lượng giao dịch có thể được thực hiện trong một lượng thời gian đã cho, cũng có nghĩa là hiệu suất sử dụng bộ xử lý và đĩa tăng lên.

- Có thể có sự trộn lẫn các giao dịch đang chạy trong hệ thống, cái thì dài cái thì ngắn. Nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến trước hoàn tất, mà điều đó dẫn đến một sự trì hoãn không lường trước được trong việc chạy một giao dịch. Nếu các giao dịch đang hoạt động trên các phần khác nhau của CSDL, sẽ tốt hơn nếu ta cho chúng chạy đồng thời, chia sẻ các chu kỳ CPU và truy xuất đĩa giữa chúng. Thực hiện cạnh tranh làm giảm sự trì hoãn không lường trước trong việc chạy các giao dịch, đồng thời làm giảm thời gian đáp ứng trung bình: Thời gian để một giao dịch được hoàn tất sau khi đã được đệ trình.

Động cơ để sử dụng thực hiện cạnh tranh trong CSDL cũng giống như động cơ để thực hiện đa chương trong hệ điều hành. Khi một vài giao dịch chạy đồng thời, tính nhất quán CSDL có thể bị phá huỷ cho dù mỗi giao dịch là đúng. Một giải pháp để giải quyết vấn đề này là sử dụng định thời. Hệ CSDL phải điều khiển sự trao đổi giữa các giao dịch cạnh tranh để ngăn ngừa chúng phá huỷ sự nhất quán của CSDL. Các cơ chế cho điều đó được gọi là sơ đồ điều khiển cạnh tranh (concurrency-control scheme).

Xét hệ thống nhà băng đơn giản, nó có một số tài khoản và có một tập hợp các giao dịch, chúng truy xuất, cập nhật các tài khoản này. Giả sử T1 và T2 là hai giao dịch chuyển khoản từ một tài khoản sang một tài khoản khác. Giao dịch T1 chuyển 50$ từ tài khoản A sang tài khoản B và được xác định như sau:

T1 : Read(A);

A:=A-50;

Write(A);

Read(B);

B:=B+50;

Write(B);

figure IV-

Giao dịch T2 chuyển 10% số dư từ tài khoản A sang tài khoản B, và được xác định như sau:

T2 :Read(A);

Temp:=A*0.1;

A:=A-temp;

Write(A);

Read(B);

B:=B+temp;

Write(B);

figure IV-

Giả sử giá trị hiện tại của A và B tương ứng là 1000$ và 2000$. Giả sử rằng hai giao dịch này được thực hiện mỗi một tại một thời điểm theo thứ tự T1 rồi tới T2. Như vậy, dãy thực hiện này là như hình bên dưới, trong đó dãy các bước chỉ thị ở trong thứ tự thời gian từ đỉnh xuống đáy, các chỉ thị của T1 nằm ở cột trái còn các chỉ thị của T2 nằm ở cột phải:

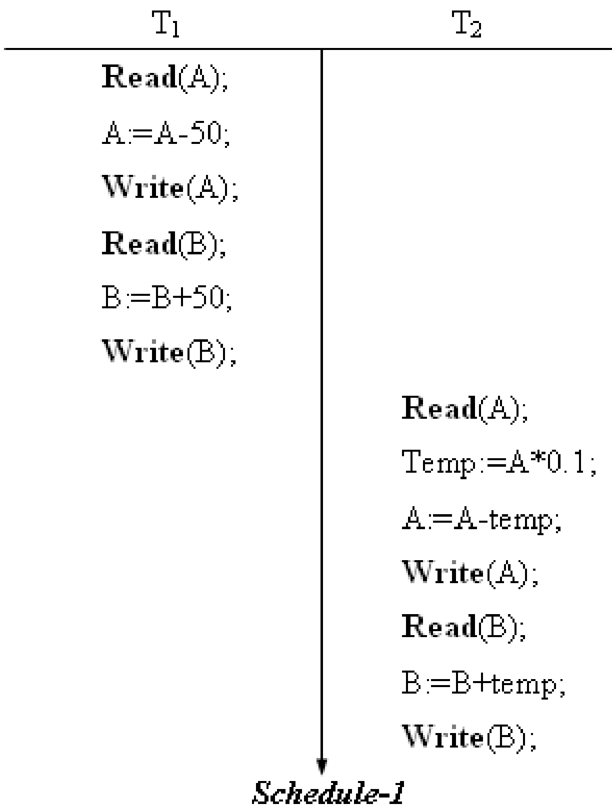


figure IV-

Giá trị sau cùng của các tài khoản A và B, sau khi thực hiện dãy các chỉ thị theo trình tự này là 855$ và 2145$ tương ứng. Như vậy, tổng giá trị của hai tài khoản này (A + B) được bảo tồn sau khi thực hiện cả hai giao dịch.

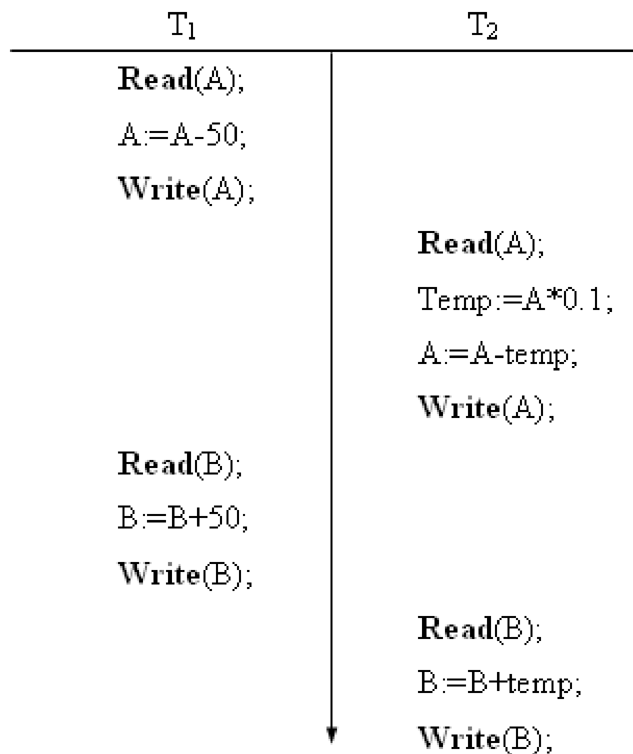
Tương tự, nếu hai giao dịch được thực hiện mỗi một tại một thời điểm song theo trình tự T2 rồi đến T1 , khi đó dãy thực hiện sẽ là:

figure IV-

Và kết quả là các giá trị cuối cùng của tài khoản A và B tương ứng sẽ là 850$ và 2150$.

Các dãy thực hiện vừa được mô tả trên được gọi là các lịch trình (schedules). Chúng biểu diễn trình tự thời gian các chỉ thị được thực hiện trong hệ thống. Một lịch trình đối với một tập các giao dịch phải bao gồm tất cả các chỉ thị của các giao dich này và phải bảo tồn thứ tự các chỉ thị xuất hiện trong mỗi một giao dịch. Ví dụ, đối với giao dịch T1 , chỉ thị Write(A) phải xuất hiện trước chỉ thị Read(B), trong bất kỳ lịch trình hợp lệ nào. Các lịch trình schedule-1 và schedule-2 là tuần tự. Mỗi lịch trình tuần tự gồm một dãy các chỉ thị từ các giao dịch, trong đó các chỉ thị thuộc về một giao dịch xuất hiện cùng nhau trong lịch trình. Như vậy, đối với một tập n giao dịch, có n! lịch trình tuần tự hợp lệ khác nhau. Khi một số giao dịch được thực hiện đồng thời, lịc trình tương ứng không nhất thiết là tuần tự. Nếu hai giao dịch đang chạy đồng thời, hệ điều hành có thể thực hiện một giao dịch trong một khoảng ngắn thời gian, sau đó chuyển đổi ngữ cảnh, thực hiện giao dịch thứ hai một khoảng thời gian sau đó lại chuyển sang thực hiện giao dịch thứ nhất một khoảng và cứ như vậy (hệ thống chia sẻ thời gian).

Có thể có một vài dãy thực hiện, vì nhiều chỉ thị của các giao dịch có thể đan xen nhau. Nói chung, không thể dự đoán chính xác những chỉ thị nào của một giao dịch sẽ được thực hiện trước khi CPU chuyển cho giao dịch khác. Do vậy, số các lịch trình có thể đối với một tập n giao dịch lớn hơn n! nhiều.



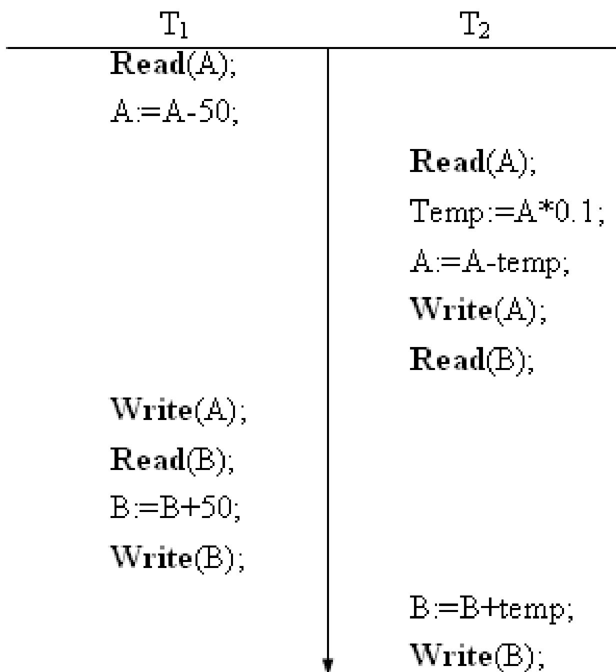
Schedule-3 --- một lịch trình cạnh tranh tương đương schedule-1

figure IV-

Không phải tất cả các thực hiện cạnh tranh cho ra một trạng thái đúng. Ví dụ schedule-4 sau cho ta một minh hoạ về nhận định này:

Sau khi thực hiện giao dịch này, ta đạt tới trạng thái trong đó giá trị cuối của A và B tương ứng là 950$ và 2100$. Trạng thái này là một trạng thái không nhất quán (A+B trước khi thực hiện giao dịch là 3000$ nhưng sau khi giao dịch là 3050$). Như vậy, nếu giao phó việc điều khiển thực hiện cạnh tranh cho hệ điều hành, sẽ có thể dẫn tới các trạng thái không nhất quán. Nhiệm vụ của hệ CSDL là đảm bảo rằng một lịch trình được phép thực hiện sẽ đưa CSDL sang một trạng thái nhất quán. Thành phần của hệ CSDL thực hiện nhiệm vụ này được gọi là thành phần điều khiển cạnh tranh (concurrency-control component). Ta có thể đảm bảo sự nhất

quán của CSDL với thực hiện cạnh tranh bằng cách nắm chắc rằng một lịch trình được thực hiện có cùng hiệu quả như một lịch trình tuần tự.



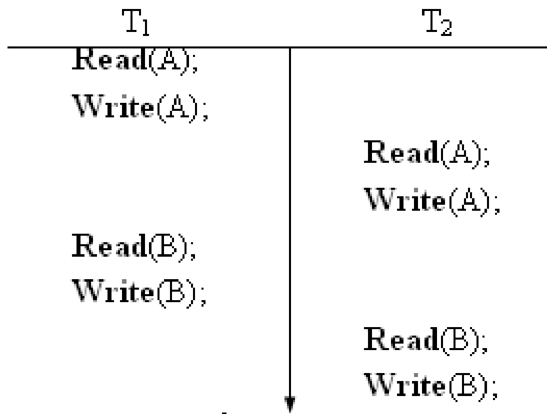
Schedule-4 --- một lịch trình cạnh tranh

figure IV-

TÍNH KHẢ TUẦN TỰ (Serializability)

Hệ CSDL phải điều khiển sự thực hiện cạnh tranh các giao dịch để đảm bảo rằng trạng thái CSDL giữ nguyên ở trạng thái nhất quán. Trước khi ta kiểm tra hệ CSDL có thể thực hiện nhiệm vụ này như thế nào, đầu tiên ta phải hiểu các lịch trình nào sẽ đảm bảo tính nhất quán và các lịch trình nào không. Vì các giao dịch là các chương trình, nên thật khó xác định các hoạt động chính xác được thực hiện bởi một giao dịch là hoạt động gì và những hoạt động nào của các giao dịch tác động lẫn nhau. Vì lý do này, ta sẽ không giải thích kiểu hoạt động mà một giao dịch có thể

thực hiện trên một hạng mục dữ liệu. Thay vào đó, ta chỉ xét hai hoạt động: Read và Write. Ta cũng giả thiết rằng giữa một chỉ thị Read(Q) và một chỉ thị Write(Q) trên một hạng mục dữ liệu Q, một giao dịch có thể thực hiện một dãy tuỳ ý các hoạt động trên bản sao của Q được lưu trú trong buffer cục bộ của giao dịch. Vì vậy ta sẽ chỉ nêu các chỉ thị Read và Write trong lịch trình, như trong biểu diễn với quy ước như vậy của schedule-3 dưới đây:



Schedule-3 (viết dưới dạng thoả thuận)

figure IV-

TUẦN TỰ XUNG ĐỘT (Conflict Serializability)

Xét lịch trình S trong đó có hai chỉ thị liên tiếp Ii và Ij của các giao dịch Ti , Tj tương ứng (i   j). Nếu Ii và Ij tham khảo đến các hạng mục dữ liệu khác nhau, ta có thể đổi chỗ Ii và Ij mà không làm ảnh hưởng đến kết quả của bất kỳ chỉ thị nào trong lịch trình. Tuy nhiên, nếu Ii và Ij tham khảo cùng một hạng mục dữ liệu Q, khi đó thứ tự của hai bước này có thể rất quan trọng. Do ta đang thực hiện chỉ các chỉ thị Read và Write, nên ta có bốn trường hợp cần phải xét sau:

1. Ii = Read(Q); Ij = Read(Q): Thứ tự của Ii và Ij không gây ra vấn đề nào, do Ti và Tj đọc cùng một giá trị Q bất kể đến thứ tự giữa Ii và

Ij.

2. Ii = Read(Q); Ij = Write(Q): Nếu Ii thực hiện trước Ij, Khi đó Ti không đọc giá trị được viết bởi Tj bởi chỉ thị Ij . Nếu Ij thực hiện trước Ii , Ti sẽ đọc giá trị của Q được viết bởi Ij , như vậy thứ tự của Ii và Ij là quan trọng.
3. Ii = Write(Q); Ij = Read(Q): Thứ tự của Ii và Ij là quan trọng do cùng lý do trong trường hợp trước.
4. Ii = Write(Q); Ij = Write(Q): Cả hai chỉ thị là hoạt động Write, thứ tự của hai chỉ thị này không ảnh hưởng đến cả hai giao dịch Ti và Tj . Tuy nhiên, giá trị nhận được bởi chỉ thị Read kế trong S sẽ bị ảnh hưởng do kết quả phụ thuộc vào chỉ thị Write được thực hiện sau cùng trong hai chỉ thị Write này. Nếu không còn chỉ thị Write nào sau Ii và Ij trong S, thứ tự của Ii vf Ij sẽ ảnh hưởng trực tiếp đến giá trị cuối của Q trong trạng thái CSDL kết quả (của lịch trình S).

Như vậy chỉ trong trường hợp cả Ii và Ij là các chỉ thị Read, thứ tự thực hiện của hai chỉ thị này (trong S) là không gây ra vấn đề.

Ta nói Ii và Ij xung đột nếu các hoạt động này nằm trong các giao dịch khác nhau, tiến hành trên cùng một hạng mục dữ liệu và có ít nhất một hoạt động là Write. Ta xét lịch trình schedule-3 như ví dụ minh hoạ cho các chỉ thị xung đột.

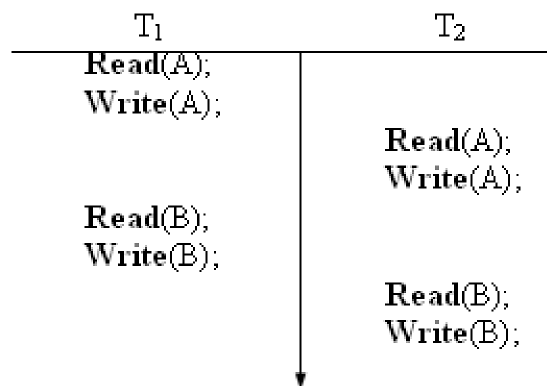


figure IV-

Chỉ thị Write(A) trong T1 xung đột với Read(A) trong T2. Tuy nhiên, chỉ thị Write(A) trong T2 không xung đột với chỉ thị Read(B) trong T1 do các

chỉ thị này truy xuất các hạng mục dữ liệu khác nhau.

Ii và Ij là hai chỉ thị liên tiếp trong lịch trình S. Nếu Ii và Ij là các chỉ thị của các giao dịch khác nhau và không xung đột, khi đó ta có thể đổi thứ tự của chúng mà không làm ảnh hưởng gì đến kết quả xử lý và như vậy ta nhận được một lịch trình mới S' tương đương với S. Do chỉ thị Write(A) của T2 không xung đột với chỉ thị Read(B) của T1, ta có thể đổi chỗ các chỉ thị này để được một lịch trình tương đương – schedule-5 dưới đây

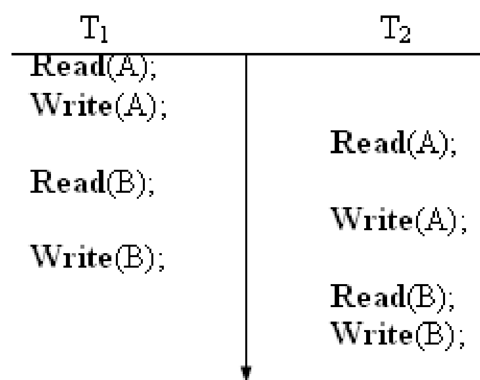
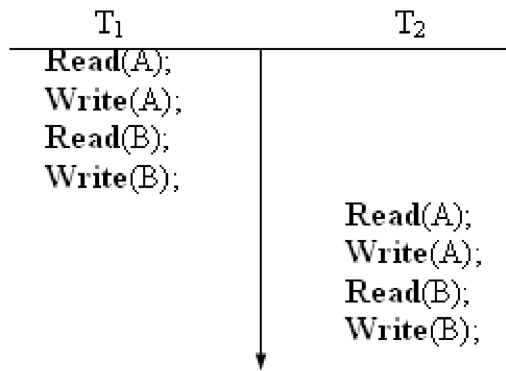


figure IV-

Ta tiếp tục đổi chỗ các chỉ thị không xung đột như sau:

- Đổi chỗ chỉ thị Read(B) của T1 với chỉ thị Read(A) của T2
- Đổi chỗ chỉ thị Write(B) của T1 với chỉ thị Write(A) của T2
- Đổi chỗ chỉ thị Write(B) của T1 với chỉ thị Read(A) của T2

Kết quả cuối cùng của các bước đổi chỗ này là một lịch trình mới (schedule-6 –lịch trình tuần tự) tương đương với lịch trình ban đầu (schedule-3):



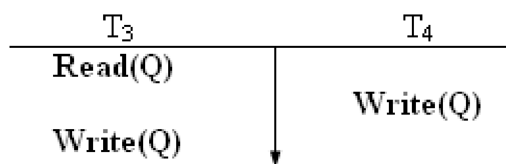
Schedule-6

figure IV-

Sự tương đương này cho ta thấy: bất chấp trạng thái hệ thống ban đầu, schedule-3 sẽ sinh ra cùng trạng thái cuối như một lịch trình tuần tự nào đó.

Nếu một lịch trình S có thể biến đổi thành một lịch trình S' bởi một dãy các đổi chỗ các chỉ thị không xung đột, ta nói S và S' là tương đương xung đột (conflict equivalent). Trong các schedule đã được nêu ở trên, ta thấy schedule-1 tương đương xung đột với schedule-3.

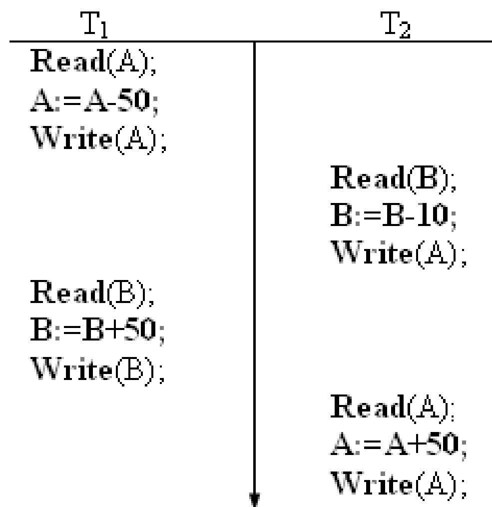
Khái niệm tương đương xung đột dẫn đến khái niệm tuần tự xung đột. Ta nói một lịch trình S là khả tuần tự xung đột (conflict serializable) nếu nó tương đương xung đột với một lịch trình tuần tự. Như vậy, schedule-3 là khả tuần tự xung đột. Như một ví dụ, lịch trình schedule-7 dưới đây không tương đương xung đột với một lịch trình tuần tự nào do vậy nó không là khả tuần tự xung đột:



Schedule-7

figure IV-

Có thể có hai lịch trình sinh ra cùng kết quả, nhưng không tương đương xung đột. Ví dụ, giao dịch T5 chuyển 10$ từ tài khoản B sang tài khoản A. Ta xét lịch trình schedule-8 như dưới đây, lịch trình này không tương đương xung đột với lịch trình tuần tự < T1, T5 > do trong lịch trình schedule-8 chỉ thị Write(B) của T5 xung đột với chỉ thị Read(B) của T1 như vậy ta không thể di chuyển tất cả các chỉ thị của T1 về trước các chỉ thị của T5 bởi việc hoán đổi liên tiếp các chỉ thị không xung đột. Tuy nhiên, các giá trị sau cùng của tài khoản A và B sau khi thực hiện lịch schedule-8 hoặc sau khi thực hiện lịch trình tuần tự <T1, T5 > là như nhau--- là 960 và 2040 tương ứng. Qua ví dụ này ta thấy cần thiết phải phân tích cả sự tính toán được thực hiện bởi các giao dịch mà không chỉ các hoạt động Read và Write. Tuy nhiên sự phân tích như vậy sẽ nặng nề và phải trả một giá tính toán cao hơn.



Schedule-8

figure IV-

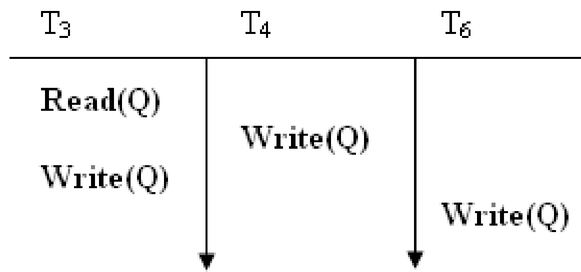
TUẦN TỰ VIEW (View Serializability)

Xét hai lịch trình S và S’, trong đó cùng một tập hợp các giao dịch tham gia vào cả hai lịch trình. Các lịch trình S và S’ được gọi là tương đương view nếu ba điều kiện sau được thoả mãn:

1. Đối với mỗi hạng mục dữ liệu Q, nếu giao dịch Ti đọc giá trị khởi đầu của Q trong lịch trình S, thì giao dịch Ti phải cũng đọc giá trị khởi đầu của Q trong lịch trình S’.
2. Đối với mỗi hạng mục dữ liệu Q, nếu giao dịch Ti thực hiện Read(Q) trong lịch trình S và giá trị đó được sản sinh ra bởi giao dịch Tj thì Ti cũng phải đọc giá trị của Q được sinh ra bởi giao dịch Tj trong S’.
3. Đối với mỗi hạng mục dữ liệu Q, giao dịch thực hiện hoạt động Write(Q) sau cùng trong lịch trình S, phải thực hiện hoạt động Write(Q) sau cùng trong lịch trình S’.

Điều kiện 1 và 2 đảm bảo mỗi giao dịch đọc cùng các giá trị trong cả hai lịch trình và do vậy thực hiện cùng tính toán. Điều kiện 3 đi cặp với các điều kiện 1 và 2 đảm bảo cả hai lịch trình cho ra kết quả là trạng thái cuối cùng của hệ thống như nhau. Trong các ví dụ trước, schedule-1 là không tương tương view với lịch trình 2 do, trong schedule-1, giá trị của tài khoản A được đọc bởi giao dịch T2 được sinh ra bởi T1 , trong khi điều này không xảy ra trong schedule-2. Schedule-1 tương đương view với schedule-3 vì các giá trị của các tài khoản A và B được đọc bởi T2 được sinh ra bởi T1 trong cả hai lịch trình.

Quan niệm tương đương view đưa đến quan niểm tuần tự view. Ta nói lịch trình S là khả tuần tự view (view serializable) nếu nó tương dương view với một lịch trình tuần tự. Ta xét lịch trình sau:



Schedule-9

figure IV-

Nó tương đương view với lịch trình tuần tự < T3, T4, T6 > do chỉ thị Read(Q) đọc giá trị khởi đầu của Q trong cả hai lịch trình và T6 thực hiện Write sau cùng trong cả hai lịch trình như vậy schedule-9 khả tuần tự view.

Mỗi lịch trình khả tuần tự xung đột là khả tuần tự view, nhưng có những lịch trình khả tuần tự view không khả tuần tự xung đột (ví dụ schedule-9).

Trong schedule-9 các giao dịch T4 và T6 thực hiện các hoạt động Write(Q) mà không thực hiện hoạt động Read(Q), Các Write dạng này được gọi là các Write mù (blind write). Các Write mù xuất hiện trong bất kỳ lịch trình khả tuần tự view không khả tuần tự xung đột.

TÍNH KHẢ PHỤC HỒI (Recoverability)

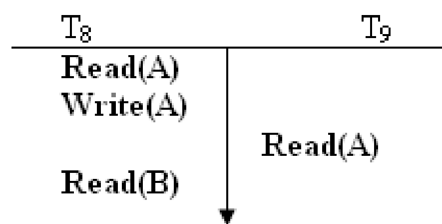
Ta đã nghiên cứu các lịch trình có thể chấp nhận dưới quan điểm sự nhát quán của CSDL với giả thiết không có giao dịch nào thất bại. Ta sẽ xét hiệu quả của thất bại giao dịch trong thực hiện cạnh tranh.

Nếu giao dịch Ti thất bại vì lý do nào đó, ta cần huỷ bỏ hiệu quả của giao dịch này để đảm bảo tính nguyên tử của giao dịch. Trong hệ thống cho phép thực hiện cạnh tranh, cũng cần thiết đảm bảo rằng bất kỳ giao dịch nào phụ thuộc vào Ti cũng phải bị bỏ. Để thực hiện sự chắc chắn

này, ta cần bố trí các hạn chế trên kiểu lịch trình được phép trong hệ thống.

LỊCH TRÌNH KHẢ PHỤC HỒI (Recoverable Schedule)

Xét lịch trình schedule-10 trong đó T9 là một giao dịch chỉ thực hiện một chỉ thị Read(A). Giả sử hệ thống cho phép T9 bàn giao (commit) ngay sau khi thực hiện chỉ thị Read(A). Như vậy T9 bàn giao trước T8 . Giả sử T8 thất bại trước khi bàn giao, vì T9 vì T9 đã đọc giá trị của hạng mục giữ liệu A được viết bởi T8 , ta phải bỏ dở T9 để đảm bảo tính nguyên tử giao dịch. Song T9 đã được bàn giao và không thể bỏ dở được. Ta có tình huống trong đó không thể khôi phục đúng sau thất bại của T8 .



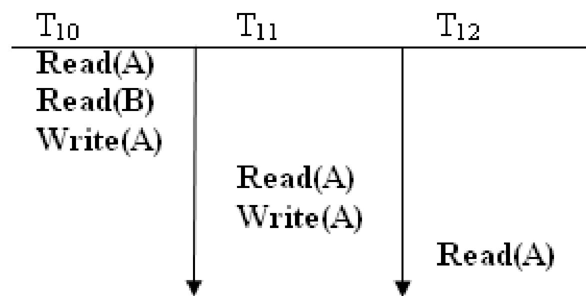
Schedule-10

figure IV-

Lịch trình schedule-10 là một ví dụ về lịch trình không phục hồi được và không dược phép. Hầu hết các hệ CSDL đòi hỏi tất cả các lịch trình phải phục hồi được. Một lịch trình khả phục hồi là lịch trình trong đó, đối với mỗi cặp giao dịch Ti , Tj , nếu Tj đọc hạng mục dữ liệu được viết bởi Ti thì hoạt động bàn giao của Tj phải xảy ra sau hoạt động bàn giao của Ti .

LỊCH TRÌNH CASCADELESS (Cascadeless Schedule)

Ngay cả khi lịch trình là khả phục hồi, để phục hồi đúng sau thất bại của một giao dịch Ti ta phải cuộn lại một vài giao dịch. Tình huống như thế xảy ra khi các giao dịch đọc dữ liệu được viết bởi Ti . Ta xét lich trình schedule-11 sau



Schedule-11

figure IV-

Giao dịch T10 viết một giá trị được đọc bởi T11 . Giao dịch T12 đọc một giá trị được viết bởi T11 . Giả sử rằng tại điểm này T10 thất bại. T10 phải cuộn lại, do T11 phụ thuộc vào T10 nên T11 cũng phải cuộn lại và cũng như vậy với T12. Hiện tượng trong đó một giao dịch thất bại kéo theo một dãy các giao dịch phải cuộn lại được gọi là sự cuộn lại hàng loạt (cascading rollback).

Cuộn lại hàng loạt dẫn đến việc huỷ bỏ một khối lượng công việc đáng kể. Phải hạn chế các lịch trình để việc cuộn lại hàng loạt không thể xảy ra. Các lịch trình như vậy được gọi là các lịch trình cascadeless. Một lịch trình cascadeless là một lịch trình trong đó mỗi cặp giao dịch Ti , Tj nếu Tj đọc một hạng mục dữ liệu được viết trước đó bởi Ti, hoạt động bàn giao của Ti phải xuất hiện trước hoạt động đọc của TJ . Một lịch trình cascadeless là khả phục hồi.

THỰC THI CÔ LẬP (Implementation of Isolation)

Có nhiều sơ đồ điều khiển cạnh tranh có thể được sử dụng để đảm bảo các tính chất một lịch trình phải có (nhằm giữ CSDL ở trạng thái nhất quán, cho phép quản lý các giao dịch ...), ngay cả khi nhiều giao dịch thực hiện cạnh tranh, chỉ các lịch trình có thể chấp nhận được sinh ra, bất kể hệ điều hành chia sẻ thời gian tài nguyên như thế nào giữa các giao dịch.

Như một ví dụ, ta xét một sơ đồ điều khiển cạnh tranh sau: Một giao dịch tậu một chốt (lock) trên toàn bộ CSDL trước khi nó khởi động và tháo chốt khi nó đã bàn giao. Trong khi giao dịch giữ chốt không giao dịch nào khác được phép tậu chốt và như vậy phải chờ đến tận khi chốt được tháo. Trong đối sách chốt, chỉ một giao dịch được thực hiện tại một thời điểm và như vậy chỉ lịch trình tuần tự được sinh ra. Sơ đồ điều khiển cạnh tranh này cho ra một hiệu năng cạnh tranh nghèo nàn. Ta nói nó cung cấp một bậc cạnh tranh nghèo (poor degree of concurrency).

Mục đích của các sơ đồ điều khiển cạnh tranh là cung cấp một bậc canh tranh cao trong khi vẫn đảm bảo các lịch trình được sinh ra là khả tuần tự xung đột hoặc khả tuần tự view và cascadeless.

ĐỊNH NGHĨA GIAO DỊCH TRONG SQL

Chuẩn SQL đặc tả sự bắt đầu một giao dịch một cánh không tường minh. Các giao dịch được kết thúc bởi một trong hai lệnh SQL sau:

- Commit work bàn giao giao dịch hiện hành và bắt đầu một giao dịch mới
- Rollback work gây ra sự huỷ bỏ giao dịch hiện hành

Từ khoá work là chọn lựa trong cả hai lệnh. Nếu một chương trình kết thúc thiếu cả hai lệnh này, các cập nhật hoặc được bàn giao hoặc bị cuộn lại là các sự thực hiện phụ thuộc.

Chuẩn cũng đặc tả hệ thống phải đảm bảo cả tính khả tuần tự và tính tự do từ việc cuộn lại hàng loạt. Định nghĩa tính khả tuần tự được ding bởi chuẩn là một lịch trình phải có cùng hiệu quả như một lịch trình tuần tự như vậy tính khả tuần tự xung đột và view đều được chấp nhận.

Chuẩn SQL-92 cũng cho phép một giao dịch đặc tả nó có thể được thực hiện theo một cách mà có thể làm cho nó trở nên không khả tuần tự với sự tôn trọng các giao dịch khác. Ví dụ, một giao dịch có thể hoạt động ở mức Read uncommitted, cho phép giao dịch đọc các mẩu tin them chí nếu chúng không được bàn giao. Đặc điểm này được cung cấp cho các giao dich dài các kết quả của chúng không nhất thiết phải chính xác. Ví dụ, thông tin xấp xỉ thường là đủ cho các thống kê được dùng cho tối ưu hoá vấn tin.

Các mức nhất quán được đặc tả trong SQL-92 là:

- Serializable : mặc nhiên
- Repeatable read : chỉ cho phép đọc các record đã được bàn giao, hơn nữa yêu cầu giữa hai Read trên một record bởi một giao dịch không một giao dịch nào khác được phép cập nhật record này. Tuy nhiên, giao dịch có thể không khả tuần tự với sự tôn trọng các giao dịch khác. Ví dụ, khi tìm kiếm các record thoả mãn các điều kiện nào đó, một giao dịch có thể tìm thấy một vài record được xen bởi một giao dịch đã bàn giao,
- Read committed: Chỉ cho phép đọc các record đã được bàn giao, nhưng không có yêu cầu thêm trên các Read khả lặp. Ví dụ, giữa hai Read của một record bởi một giao dịch, các mẩu tin có thể được cập nhật bởi các giao dịch đã bàn giao khác.
- Read uncommitted: Cho phép đọc cả các record chưa được bàn giao. Đây là mức nhất quán thấp nhất được phép trong SQL-92.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ

Khi thiết kế các sơ đồ điều khiển cạnh tranh, ta phải chứng tỏ rằng các lịch trình được sinh ra bởi sơ đồ là khả tuần tự. Để làm điều đó, trước tiên ta phải biết làm thế nào để xác định, với một lịch trình cụ thể đã cho, có là khả tuần tự hay không.

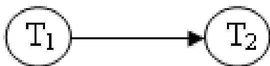
KIỂM THỬ TÍNH KHẢ TUẦN TỰ XUNG ĐỘT

Giả sử S là một lịch trình. Ta xây dựng một đồ thị định hướng, được gọi là đồ thị trình tự (precedence graph), từ S. Đồ thị gồm một cặp (V, E) trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh bao gồm tất cả các giao dịch tham gia vào lịch trình. Tập các cung bao gồm tất cả các cung dạng Ti   Tj sao cho một trong các điều kiện sau được thoả mãn:

1. Ti thực hiện Write(Q) trước Tj thực hiện Read(Q).
2. Ti thực hiện Read(Q) trước khi Tj thực hiện Write(Q).
3. Ti thực hiện Write(Q) trước khi Tj thực hiện Write(Q).

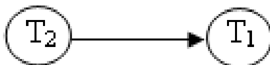
Nếu một cung Ti   Tj tồn tại trong đồ thị trình tự, thì trong bất kỳ lịch trình tuần tự S' nào tương đương với S, Ti phải xuất hiện trước Tj .

Đồ thị trình tự đối với schedule-1 là:



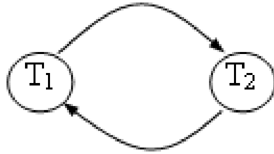
vì tất cả các chỉ thị của T1 được thực hiện trước chỉ thị đầu tiên của T2

Đồ thị trình tự đối với schedule-2 là:

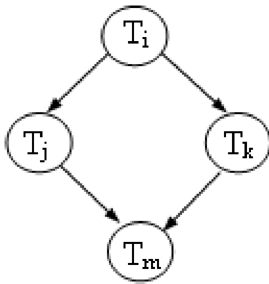


vì tất cả các chỉ thị của T2 được thực hiện trước chỉ thị đầu tiên của T1

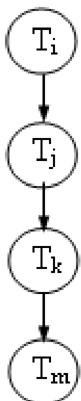
Đồ thị trình tự đối với schedule-4 chứa các cung T1   T2 vì T1 thực hiện Read(A) trước T2 thực hiện Write(A). Nó cũng chứa cung T2   T1 vì T2 thực hiện Read(B) trước khi T1 thực hiện Write(B):



Nếu đồ thị trình tự đối với S có chu trình, khi đó lịch trình S không là khả tuần tự xung đột. Nếu đồ thị không chứa chu trình, khi đó lịch trình S là khả tuần tự xung đột. Thứ tự khả tuần tự có thể nhận được thông qua sắp xếp topo (topological sorting), nó xác định một thứ tự tuyến tính nhất quán với thứ tự bộ phận của đồ thị trình tự. Nói chung, có một vài thứ tự tuyến tính có thể nhận được qua sắp xếp topo. Ví dụ, đồ thị sau:



Có hai thứ tự tuyến tính chấp nhận được là:



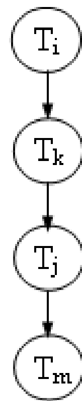


figure IV-

Như vậy, để kiểm thử tính khả tuần tự xung đột, ta cần xây dựng đồ thị trình tự và gọi thuật toán phát hiện chu trình. Ta nhận được một sơ đồ thực nghiệm để xác định tính khả tuần tự xung đột. Như ví dụ, schedule-1 và schedule-2, đồ thị trình tự của chúng không có chu trình, do vậy chúng là các chu trình khả tuần tự xung đột, trong khi đồ thị trình tự của schedule-4 chứa chu trình do vậy nó không là khả tuần tự xung đột.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ VIEW

Ta có thể sửa đổi phép kiểm thử đồ thị trình tự đối với tính khả tuần tự xung đột để kiểm thử tính khả tuần tự view. Tuy nhiên, phép kiểm thử này phải trả giá cao về thời gian chạy.

Xét lịch trình schedule-9, nếu ta tuân theo quy tắc trong phép kiểm thử tính khả tuần tự xung đột để tạo đồ thị trình tự, ta nhận được đồ thị sau:

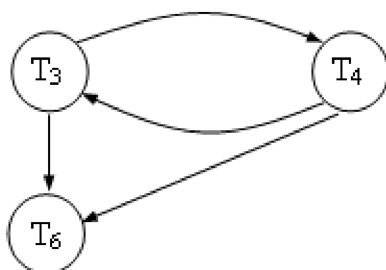


figure IV-

Đồ thị này có chu trình, do vậy schedule-9 không là khả tuần tự xung đột. Tuy nhiên, đã đã thấy nó là khả tuần tự view (do nó tương đương với lịch trình tuần tự < T3, T4 , T6 >). Cung T3 T4 không được xen vào đồ thị vì các giá trị của hạng mục Q được sản sinh bởi T3 và T4 không được dùng bởi bất kỳ giao dịch nào khác và T6 sản sinh ra giá trị cuối mới của Q. Các chỉ thị Write(Q) của T3 và T4 được gọi là các Write vô dụng (Useless Write). Điều trên chỉ ra rằng không thể sử dụng đơn thuần sơ đồ đồ thị

trình tự dể kiểm thử tính khả tuần tự view. Cần thiết phát triển một sơ đồ cho việc quyết định cung nào là cần phải xen vào đồ thị trình tự.

Xét một lịch trình S. Giả sử giao dịch Tj đọc hạng mục dữ liệu Q được viết bởi Ti . Rõ ràng là nếu S là khả tuần tự view, khi đó, trong bất kỳ lịch trình tuần tự S' tương đương với S, Ti phải đi trước Tj . Bây giờ giả sử rằng, trong lịch trình S, giao dịch Tk thực hiện một Write(Q), khi đó, trong lịch trình S', Tk phải hoặc đi trước Ti hoặc đi sau Tj . Nó không thể xuất hiện giữa Ti và Tj vì như vậy Tj không đọc giá trị của Q được viết bởi Ti và như vậy S không tương đương view với S'. Các ràng buộc này không thể biểu diễn được trong thuật ngữ của mô hình đồ thị trình tự đơn giản được nêu lên trước đây. Như trong ví dụ trước, khó khăn nảy sinh ở chỗ ta biết một trong hai cung Tk   Ti và Tj   Tk phải được xen vào đồ thị nhưng ta chưa tạo được quy tắc để xác định sự lựa chọn thích hợp. Để tạo ra quy tắc này, ta cần mở rộng đồ thị định hướng để bao hàm các cung gán nhãn, ta gọi đồ thị như vậy là đồ thị trình tự gán nhãn (Label precedence graph). Cũng như trước đây, các nút của đồ thị là tất cả các giao dịch tham gia vào lịch trình. Các quy tắc xen cung gán nhãn được diễn giải như sau:

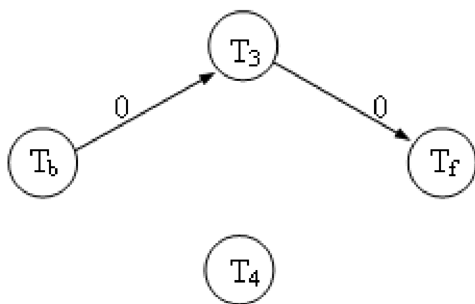
Giả sử S là lịch trình gồm các giao dịch { T1 , T2 , ... , Tn }. Tb và Tf là hai giao dịch giả: Tb phát ra Write(Q) đối với mỗi Q được truy xuất trong S, Tf phát ra Read(Q) đối với mỗi Q được truy xuất trong S. Ta xây dựng lich trình mới S' từ S bằng cách xen Tb ở bắt đầu của S và Tf ở cuối của S. Đồ thị trình tự gán nhãn đố với S' được xây dung dựa trên các quy tắc:

1. Thêm cung Ti   0 Tj , nếu Tj đọc giá trị của hạng mục dữ liệu Q được viết bởi Ti
2. Xoá tất cả các cung liên quan tới các giao dịch vô dụng. Một giao dịch Ti được gọi là vô dụng nếu không có con đường nào trong đồ thị trình tự dẫn từ Ti đến Tf .
3. Đối với mỗi hạng mục dữ liệu Q sao cho Tj đọc giá trị của Q được viết bởi Ti và Tk thực hiện Write(Q), Tk   Tb tiến hành các bước sau

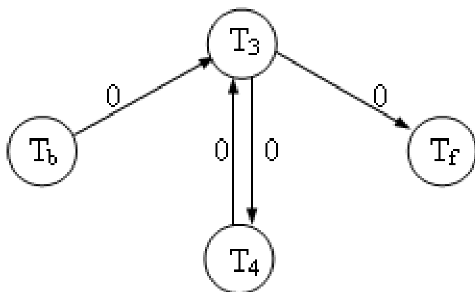
    - Nếu Ti = Tb và Tj   Tf, khi đó xen cung Tj   0 Tk vào đồ thị trình tự gán nhãn

- Nếu Ti   Tb và Tj = Tf khi đó xen cung Tk 0 Ti vào đồ thị trình tự gán nhãn
- Nếu Ti   Tb và Tj   Tf khi đó xen cả hai cung Tk p Ti và Tj p Tk vào đồ thị trình tự gán nhãn, trong đó p là một số nguyên duy nhất lớn hơn 0 mà chưa được sử dụng trước đó để gán nhãn cung.

Quy tắc 3c phản ánh rằng nếu Ti viết hạng mục dữ liệu được đọc bởi Tj thì một giao dịch Tk viết cùng hạng mục dữ liệu này phải hoặc đi trước Ti hoặc đi sau Tj . Quy tắc 3a và 3b là trường hợp đặc biệt là kết quả của sự kiện Tb và Tf cần thiết là các giao dịch đầu tiên và cuối cùng tương ứng. Như một ví dụ, ta xét schedule-7. Đồ thị trình tự gán nhãn của nó được xây dựng qua các bước 1 và 2 là:



Đồ thị sau cùng của nó là (cung T3   T4 là kết quả của 3a, cung T4   T3 là kết quả của 3b) :



Đồ thị trình tự gán nhãn của schedule-9 là:

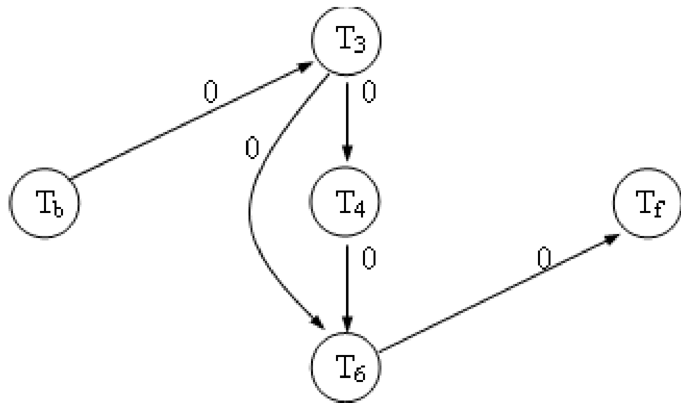


figure IV-

Cuối cùng, ta xét lịch trình schedule-10:

T_3	T_3	T_7
Read(Q)		
	Write(Q)	
		Read(Q)
Write(Q)		
		Write(Q)

Đồ thị trình tự gán nhãn của schedule-10 là:

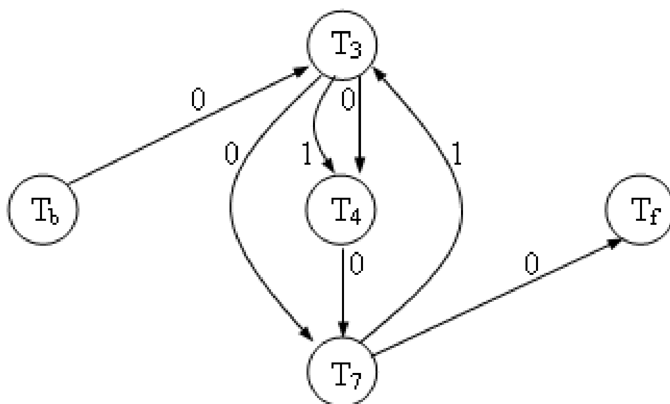


figure IV-

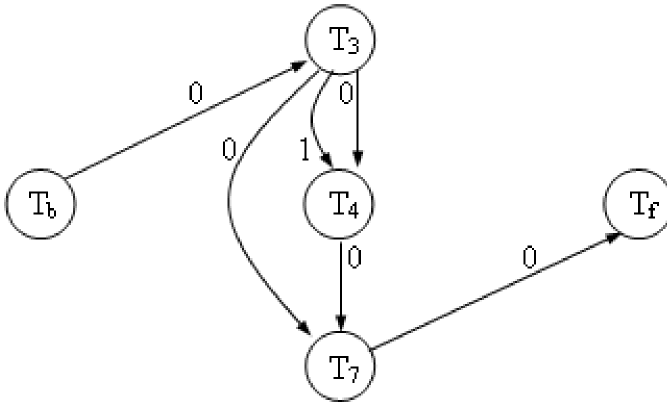
Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình tối tiểu : T3   T4 T3

Đồ thị trình tự gán nhãn của schedule-10 chứa chu trình tối tiểu: T3   T1 T3

Đồ thị trình tự gán nhãn của schedule-9 không chứa chứa chu trình nào.

Nếu đồ thị trình tự gán nhãn không chứa chu trình, lịch trình tương ứng là khả tuàn tự view, như vậy schedule-9 là khả tuần tự view. Tuy nhiên, nếu đồ thị chứa chu trình, điều kiện này không kéo theo lịch trình tương ứng không là khả tuần tự view. Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình và lịch trình này không là khả tuần tự view. Bên cạnh đó, lịch trình schedule-10 là khả tuần tự view, nhưng đồ thị trình tự gán nhãn của nó có chứa chu trình. Bây giờ ta giả sử rằng có n cặp cung tách biệt, đó là do ta đã áp dụng n lần quy tắc 3c trong sự xây dựng đồ thị trình tự. Khi đó có 2n đồ thị khác nhau, mỗi một chứa đúng một cung trong mỗi cặp. Nếu một đồ thị nào đó trong các đồ thị này là phi chu trình, khi đó lịch trình tương ứng là khả tuần tự view. Thuật toán này đòi hỏi một phép kiểm thử vét cạn các đồ thị riêng biệt, và như vậy thuộc về lớp vấn đề NP-complet !!!

Ta xét đồ thị schedule-10. nó có đúng một cặp tách biệt. Hai đồ thị triên biệt là:



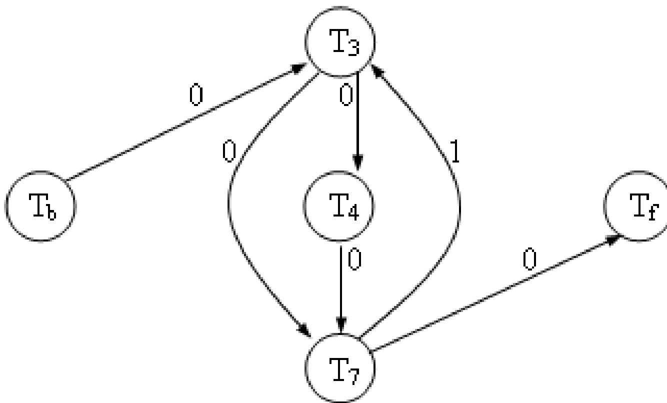


figure IV-

Đồ thị thứ nhất không chứa chu trình, do vậy lịch trình là khả tuần tự view.

BÀI TẬP CHƯƠNG IV

IV.1 Liệt kê các tính chất ACID. Giải thích sự hữu ích của mỗi một trong chúng.

IV.2Trong khi thực hiện, một giao dịch trải qua một vài trạng thái đến tận khi nó bàn giao hoặc bỏ dở. Liệt kê tất cả các dãy trạng thái có thể giao dịch có thể trải qua. Giải thích tại sao mỗi bắc cầu trạng thái có thể xảy ra.

IV.3 Giải thích sự khác biệt giữa lịch trình tuần tự (Serial schedule) và lịch trình khả tuần tự (Serializable schedule).

IV.4Xét hai giao dịch sau:

T1 : Read(A);

Read(B);

If A=0 then B:=B+1;

Write(B).

T2 :Read(B);

Read(A);

If B=0 then A:=A+1;

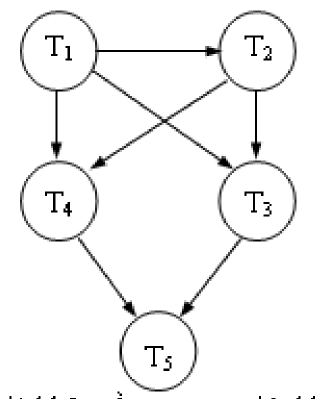
Write(A).

Giả thiết yêu cầu nhất quán là A=0 V B=0 với A=B=0 là các giá trị khởi đầu

- Chứng tỏ rằng mỗi sự thự hiện tuần tự bao gồm hai giao dịch này bảo tồn tính nhất quán của CSDL.
- Nêu một sự thực hiện cạnh tranh của T1 và T2 sinh ra một lịch trình không khả tuần tự.
- Có một sự thực hiện cạnh tranh của T1 và T2 sinh ra một lịch trình khả tuần tự không ?

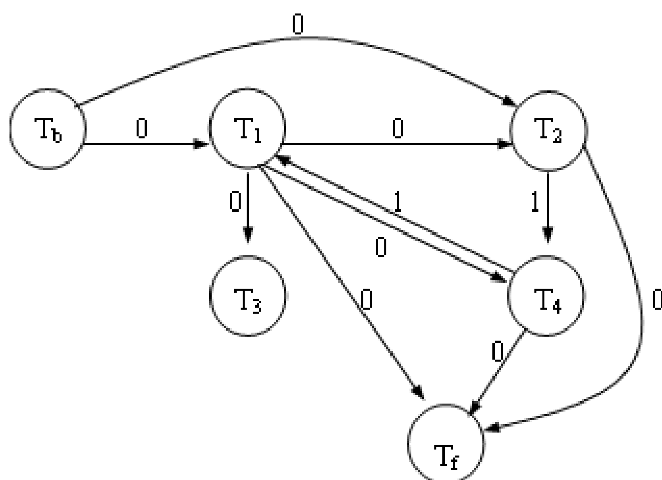
IV.5Do một lịch trình khả tuần tự xung đột là một lịch trình khả tuần tự view. Tại sao ta lại nhấn mạnh tính khả tuần tự xung đột hơn tính khả tuần tự view?

IV.6Xét đồ thị trình tự sau:



Lịch trình tương ứng là khả tuần tự xung đột không? Giải thích

IV.7Xét đồ thị trình tự gán nhãn sau:



Lịch trình tương ứng là khả tuần tự view không? Giải thích.

IV.8Lịch trình khả phục hồi là gì? Tại sao cần thiết tính khả phục hổi của một lịch trình?

IV.9Lịch trình cascadeless là gì? Tại sao cần thiết tính cascadeless của lịch trình?

Điều khiển cạnh tranh
MỤC ĐÍCH Một trong các tính chất cơ bản của một giao dịch là tính cô lập. Khi một vài giao dịch thực hiện một cách cạnh tranh trong CSDL, tính cô lập có thể không được bảo tồn. Đối với hệ thống, cần phải điều khiển sự trao đổi giữa các giao dịch cạnh tranh; sự điều khiển này được thực hiện thông qua một trong tập hợp đa dạng các cơ chế được gọi là sơ đồ điều khiển cạnh tranh. Các sơ đồ điều khiển cạnh tranh được xét trong chương này được dựa trên tính khả tuần tự. Trong chương này ta cũng xét sự quản trị các giao dịch thực hiện cạnh canh nhưng không xét đến sự cố hỏng hóc. YÊU CẦU Hiểu các khái niệm Hiểu các kỹ thuật điều khiển cạnh tranh: - Các kỹ thuật dựa trên chốt (lock) - Các kỹ thuật dựa trên tem thời gian - Các kỹ thuật hỗp hợp Hiểu nguyên lý của các kỹ thuật này Hiểu các kỹ thuật điều khiển deadlock

GIAO THỨC DỰA TRÊN CHỐT

Một phương pháp để đảm bảo tính khả tuần tự là yêu cầu việc truy xuất đến hạng mục dữ liệu được tiến hành theo kiểu loại trừ tương hỗ; có nghĩa là trong khi một giao dịch đang truy xuất một hạng mục dữ liệu, không một giao dịch nào khác có thể sửa đổi hạng mục này. Phương pháp chung nhất được dùng để thực thi yêu cầu này là cho phép một giao dịch truy xuất một hạng mục dữ liệu chỉ nếu nó đang giữ chốt trên hạng mục dữ liệu này.

CHỐT (Lock)

Có nhiều phương thức chốt hạng mục dữ liệu. Ta hạn chế việc nghiên cứu trên hai phương thức:

1. Shared. Nếu một giao dịch Ti nhận được một chốt ở phương thức shared (ký hiệu là S) trên hạng mục Q, khi đó Ti có thể đọc, nhưng không được viết Q.
2. Exclusive. Nếu một giao dịch Ti nhận được một chốt ở phương thức Exclusive (ký hiệu là X), khi đó Ti có thể cả đọc lẫn viết Q.

Ta yêu cầu là mỗi giao dịch đòi hỏi một chốt ở một phương thức thích hợp trên hạng mục dữ liệu Q, phụ thuộc vào kiểu hoạt động mà nó sẽ thực hiện trên Q. Giả sử một giao dịch Ti đồi hỏi một chốt phương thức A trên hạng mục Q mà trên nó giao dich Tj (Tj   Ti) hiện đang giữ một chốt phương thức B. Nếu giao dịch Ti có thể được cấp một chốt trên Q ngay, bất chấp sự hiện diện của chốt phương thức B, khi đó ta nói phương thức A tương thích với phương thức B. Một hàm như vậy có thể được biểu diễn bởi một ma trận. Quan hệ tương thích giữa hai phương thức chốt được cho bởi ma trận comp sau:

	S	X
S	True	False
X	False	False

Comp(A, B)= true có nghĩa là các phương thức A và B tương thích.

figure V- 1

Các chốt phương thức shared có thể được giữ đồng thời trên một hạng mục dữ liệu. Một chốt exclusive đến sau phải chờ đến tận khi tất cả các chốt phương thức shared đến trước được tháo ra.

Một giao dịch yêu cầu một chốt shared trên hạng mục dữ liệu Q bằng cách thực hiện chỉ thị lock-S(Q), yêu cầu một chốt exclusive thông qua chỉ thị lock-X(Q). Một hạng mục dữ liệu Q có thể được tháo chốt thông qua chỉ thị unlock(Q).

Để truy xuất một hạng mục dữ liệu, giao dịch Ti đầu tiên phải chốt hạng mục này. Nếu hạng mục này đã bị chốt bởi một giao dịch khác ở phương thức không tương thích, bộ điều khiển cạnh tranh sẽ không cấp chốt cho đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được tháo. Như vậy Ti phải chờ đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được giải phóng.

Giao dịch Ti có thể tháo chốt một hạng mục dữ liệu mà nó đã chốt trước đây. Một giao dịch cần thiết phải giữ một chốt trên một hạng mục dữ liệu chừng nào mà nó còn truy xuất hạng mục này. Hơn nữa, đối với một

giao dịch việc tháo chốt ngay sau truy xuất cuối cùng đến hạng mục dữ liệu không luôn luôn là điều mong muốn vì như vậy tính khả tuần tự có thể không được đảm bảo. Để minh hoạ cho tình huống này, ta xét ví dụ sau: A và B là hai tài khoản có thể được truy xuất bởi các giao dịch T1 và T2 . Giao dịch T1 chuyển 50$ từ tài khoản B sang tài khoản A và đươch xác định như sau:

```
T1 :    Lock-X(B);
        Read(B);
        B:=B-50;
        Write(B);
        Unlock(B);
        Lock-X(A);
        Read(A);
        A:=A+50;
        Write(A);
        Unlock(A);
```

figure V- 1

Giao dịch T2 hiển thị tổng số lượng tiền trong các tài khoản A và B (A + B) và được xác định như sau;

```
T2 :    Lock-S(A);
        Read(A);
        Unlock(A);
        Lock-S(B);
        Read(B);
        Unlock(B);
        Display(A+B);
```

figure V- 1

Giả sử giá trị của tài khoản A và B tương ứng là 100$ và 200$. Nếu hai giao dịch này thực hiện tuần tự, hoặc theo thứ tự T1, T2 hoặc theo thứ tự T2 , T1 , và khi dó T2 sẽ hiển thị giá trị 300$. Tuy nhiên nếu các giao dịch này thực hiện cạnh tranh, giả sử theo lịch trình schedule-1, trong trường hợp như vậy giao dịch T2 sẽ hiển thị giá trị 250$ --- một kết quả không đúng. Lý do của sai lầm này là do giao dịch T1 đã tháo chốt hạng mục B quá sớm và T2 đã tham khảo một trạng thái không nhất quán !!!

Lịch trình schedule 1 bày tỏ các hành động được thực hiện bởi các giao dịch cũng như các thời điểm khi các chốt được cấp bởi bộ quản trị điều khiển cạnh tranh. Giao dịch đưa ra một yêu cầu chốt không thể thực hiện hành động kế của mình đến tận khi chốt được cấp bởi bộ quản trị điều khiển cạnh tranh; do đó, chốt phải được cấp trong khoảng thời gian giữa hoạt động yêu cầu chốt và hành động sau của giao dịch. Sau này ta sẽ luôn giả thiết chốt được cấp cho giao dịch ngay trước hành động kế và như vậy ta có thể bỏ qua cột bộ quản trị điều khiển cạnh tranh trong bảng

T_1	T_2	Bộ quản trị điều khiển cạnh tranh
Lock-X(B)		
		Grant-X(B,T_1)
Read(B)		
B:=B-50		
Write(B)		
Unlock(B)		
	Lock-S(A)	
		Grant-S(A,T_2)
	Read(A)	
	Unlock(A)	
	Lock-S(B)	
		Grant-S(B,T_2)
	Read(B)	
	Unlock(B)	
	Display(A+B)	
Lock-X(A)		
		Grant-X(A,T_1)
Read(A)		
A:=A+50		
Write(A)		
Unlock(A)		

Schedule-1

figure V- 1

Bây giờ giả sử rằng tháo chốt bị làm trễ đến cuối giao dịch. Giao dịch T3 tương ứng với T1 với tháo chốt bị làm trễ được định nghĩa như sau:

```
T3 :    Lock-X(B);
        Read(B);
        B:=B-50;
        Write(B);
        Lock-X(A);
        Read(A);
        A:=A+50;
        Write(A);
        Unlock(B);
        Unlock(A);
```

figure V- 1

Giao dịch T4 tương ứng với T2 với tháo chốt bị làm trễ được xác định như sau:

```
T4 :    Lock-S(A);
        Read(A);
        Lock-S(B);
        Read(B);
        Display(A+B);
        Unlock(A);
        Unlock(B);
```

figure V- 1

Các lịch trình có thể trên T3 và T4 không để cho T4 hiển thị trạng thái không nhất quán.

Tuy nhiên, sử dụng chốt có thể dẫn đến một tình huống không mong đợi. Ta hãy xét lịch trình bộ phận schedule-2 trên T3 và T4 sau:

T_3	T_4
Lock-X(B)	
Read(B)	
B:=B-50	
Write(B)	
	Lock-S(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

Schedule-2

figure V- 1

Do T3 giữ một chốt phương thức Exclusive trên B, nên yêu cầu một chốt phương thức shared của T4 trên B phải chờ đến khi T3 tháo chốt. Cũng vậy, T3 yêu cầu một chốt Exclusive trên A trong khi T4 đang giữ một chốt shared trên nó và như vậy phải chờ. Ta gặp phải tình huống trong đó T3 chờ đợi T4 đồng thời T4 chờ đợi T3 -- một sự chờ đợi vòng tròn -- và như vậy không giao dịch nào có thể tiến triển. Tình huống này được gọi là deadlock (khoá chết). Khi tình huống khoá chết xảy ra hệ thống buộc phải cuộn lại một trong các giao dịch. Mỗi khi một giao dịch bị cuộn lại, các hạng mục dữ liệu bị chốt bởi giao dịch phải được tháo chốt và nó trở nên sẵn có cho giao dịch khác, như vậy các giao dịch này có thể tiếp tục được sự thực hiện của nó.

Nếu ta không sử dụng chốt hoặc tháo chốt hạng mục dữ liệu ngay khi có thể sau đọc hoặc viết hạng mục, ta có thể rơi vào trạng thái không nhất quán. Mặt khác, nếu ta không tháo chốt một hạng mục dữ liệu trước khi yêu cầu một chốt trên một hạng mục khác, dealock có thể xảy ra. Có các phương pháp tránh dealock trong một số tình huống, tuy nhiên nói chung dealock là khó tránh khi sử dụng chốt nếu ta muốn tránh trạng thái không nhất quán. Dealock được ưa thích hơn trạng thái không nhất quán vì chúng có thể điều khiển được bằng cách cuộn lại các giao dịch trong khi đó trạng thái không nhất quán có thể dẫn đến các vấn đề thực tế mà hệ CSDL không thể điều khiển.

Ta sẽ yêu cầu mỗi giao dịch trong hệ thống tuân theo một tập các quy tắc , được gọi là giao thức chốt (locking protocol), chỉ định khi một giao dịch

có thể chốt và tháo chốt mỗi một trong các hạng mục dự liệu. Giao thức chốt hạn chế số các lịch trình có thể. Tập các lịch trình như vậy là một tập con thực sự của tập tất cả các lịch trình khả tuần tự có thể.

Xét { T0 , T1 , ..., Tn } một tập các giao dịch tham gia vào lịch trình S. Ta nói Ti đi trước Tj trong S, và được viết là Ti   Tj , nếu tồn tại một hạng mục dữ liệu Q sao cho Ti giữ chốt phương thức A trên Q , Tj giữ chốt phương thức B trên Q muộn hơn và comp(A,B) = false. Nếu Ti   Tj , thì Ti sẽ xuất hiện trước Tj trong bất kỳ lịch trình tuần tự nào.

Ta nói một lịch trình S là hợp lệ dưới một giao thức chốt nếu S là một lịch trình tuân thủ các quy tắc của giao thức chốt đó. Ta nói rằng một giao thức chốt đảm bảo tính khả tuần tự xung đột nếu và chỉ nếu đối với tất cả các lịch trình hợp lệ, quan hệ   kết hợp là phi chu trình.

CẤP CHỐT

Khi một giao dịch yêu cầu một chốt trên một hạng mục dữ liệu ở một phương thức và không có một giao dịch nào khác giữ một chốt trên cùng hạng mục này ở một phương thức xung đột, chốt có thể được cấp. Tuy nhiên, phải thận trọng để tránh kịch bản sau: giả sử T2 giữ một chốt phương thức shared trên một hạng mục dữ liệu, một giao dịch khác T1 yêu cầu một chốt phương thức exclusive cũng trên hạng mục này, rõ ràng T1 phải chờ T2 tháo chốt. Trong khi đó một giao dịch khác T3 yêu cầu một chốt phương thức shared, do yêu cầu chốt này tương thích với phương thức chốt được giữ bởi T1 nên nó được cấp cho T3. Tại thời điểm T2 tháo chốt, T1 vẫn phải chờ sự tháo chốt của T3, nhưng bây giờ lại có một giao dịch T4 yêu cầu một chốt phương thức shared và nó lại được cấp do tính tương thích và cứ như vậy, có thể T1 sẽ không bao giờ được cấp chốt mà nó yêu cầu trên hạng mục dữ liệu. Ta gọi hiện tượng này là bị chết đói (starved).

Để tránh sự chết đói của các giao dịch, việc cấp chốt được tiến hành như sau: Khi một giao dịch Ti yêu cầu một chốt trên một hạng mục dữ liệu Q

ở phương thức M, chốt sẽ được cấp nếu các điều kiện sau được thoả mãn:

1. Không có giao dịch khác đang giữ một chốt trên Q ở phương thức xung đột với M
2. Không có một giao dịch nào đang chờ được cấp một chốt trên M và đã đưa ra yêu cầu về chốt trước Ti

GIAO THỨC CHỐT HAI KỲ (Two-phase locking protocol)

Giao thức chốt hai kỳ là một giao thức đảm bảo tính khả tuần tự. Giao thức này yêu cầu mỗi một giao dịch phát ra yêu cầu chốt và tháo chốt thành hai kỳ:

1. Kỳ xin chốt (Growing phase). Một giao dịch có thể nhận được các chốt, nhưng có không thể tháo bất kỳ chốt nào
2. Kỳ tháo chốt (Shrinking phase). Một giao dịch có thể tháo các chốt nhưng không thể nhận được một chốt mới nào.

Khởi đầu, một giao dịch ở kỳ xin chốt. Giao dịch tậu được nhiều chốt như cần thiết. Mỗi khi giao dịch tháo một chốt, nó đi vào kỳ tháo chốt và nó không thể phát ra bất kỳ một yêu cầu chốt nào nữa. Các giao dich T3 và T4 là hai kỳ. Các giao dịch T1 và T2 không là hai kỳ. Người ta có thể chứng minh được giao thức chốt hai kỳ đảm bảo tính khả tuần tự xung đột, nhưng không đảm bảo tránh được dealock và việc cuộn lại hàng loạt. Cuộn lại hàng loạt có thể tránh được bởi một sự sửa đổi chốt hai kỳ được gọi là giao thức chốt hai kỳ nghiêm ngặt. Chốt hai kỳ nghiêm ngặt đòi hỏi thêm tất cả các chốt phương thức exclusive phải được giữ đến tận khi giao dịch bàn giao. Yêu cầu này đảm bảo rằng bất kỳ dữ liệu nào được viết bởi một giao dịch chưa bàn giao bị chốt trong phương thức exclusive đến tận khi giao dịch bàn giao, điều đó ngăn ngừa bất kỳ giao dịch khác đọc dữ liệu này.

Một biến thể khác của chốt hai kỳ là giao thức chốt hai kỳ nghiêm khắc. Nó đòi hỏi tất cả các chốt được giữ đến tận khi giao dịch bàn giao. Hầu

hết các hệ CSDL thực hiện chốt hai kỳ nghiêm ngặt hoặc nghiêm khắc.

Một sự tinh chế giao thức chốt hai kỳ cơ sở dựa trên việc cho phép chuyển đổi chốt: nâng cấp một chốt shared sang exclusive và hạ cấp một chốt exclusive thành chốt shared. Chuyển đổi chốt không thể cho phép một cách tuỳ tiện, nâng cấp chỉ được phép diễn ra trong kỳ xin chốt, còn hạ cấp chỉ được diễn ra trong kỳ tháo chốt. Một giao dịch thử nâng cấp một chốt trên một hạng mục dữ liệu Q có thể phải chờ. Giao thức chốt hai kỳ với chuyển đổi chốt cho phép chỉ sinh ra các lịch trình khả tuần tự xung đột. Nếu các chốt exclusive được giữ đến tận khi bàn giao, các lịch trình sẽ là cascadeless.

Ta xét một ví dụ: Các giao dịch T8 và T9 được nêu trong ví dụ chỉ được trình bày bởi các hoạt động ý nghĩa là Read và Write.

T8 : Read(A1);

Read(A2);

...

Read(An);

Write(A1).

T9 : Read(A1);

Read(A2);

Display(A1 + A2).

figure V-

Nếu ta sử dụng giao thức chốt hai kỳ, khi đó T8 phải chốt A1 ở phương thức exclusive. Bởi vậy, sự thực hiện cạnh tranh của hai giao dịch rút cuộc trở thành thực hiện tuần tự. Ta thấy rằng T8 cần một chốt exclusive trên A1 chỉ ở cuối sự thực hiện của nó, khi nó write(A1). Như vậy, T8 có thể khởi động chốt A1 ở phương thức shared, và đổi chốt này sang

phương thức exclusive sau này. Như vậy ta có thể nhận được tính cạnh tranh cao hơn, vì như vậy T8 và T9 có thể truy xuất đến A1 và A2 đồng thời.

Ta biểu thị sự chuyển đổi từ phương thức shared sang phương thức exclusive bởi upgrade và từ phương thức exclusive sang phương thức shared bởi downgrade. Upgrade chỉ được phép xảy ra trong kỳ xin chốt và downgrade chỉ được phép xảy ra trong kỳ tháo chốt. Lịch trình chưa hoàn tất dưới đây cho ta một minh hoạ về giao thức chốt hai kỳ với chuyển đổi chốt.

Chú ý rằng một giao dịch thử cập nhật một chốt trên một hạng mục dữ liệu Q có thể buộc phải chờ. Việc chờ bắt buộc này xảy ra khi Q đang bị chốt bởi giao dịch khác ở phương thức shared.

Giao thức chốt hai kỳ với chuyển đổi chốt chỉ sinh ra các lịch trình khả tuần tự xung đột, các giao dịch có thể được tuần tự hoá bởi các điểm chốt của chúng. Hơn nữa, nếu các chốt exclusive được giữ đến tận khi kết thúc giao dịch, lịch trình sẽ là cascadeless.

T_8	T_9
Lock-S(A_1)	
	Lock-S(A_2)
Lock-S(A_2)	
	Lock-S(A_2)
Lock-S(A_3)	
...	
	Unlock(A_1)
	Unlock(A_2)
UpGrade(A_1)	

figure V- 1

Ta mô tả một sơ đồ đơn giản nhưng dược sử dụng rộng rãi để sinh tự động các chỉ thị chốt và tháo chốt thích hợp cho một giao dịch: Mỗi khi giao dich T xuất ra một chỉ thị Read(Q), hệ thống sẽ xuất ra một chỉ thị Lock-S(Q) ngay trước chỉ thị Read(Q). Mỗi khi giao dịch T xuất ra một

hoạt động Write(Q), hệ thống sẽ kiểm tra xem T đã giữ một chốt shared nào trên Q hay chưa, nếu đã, nó xuất ra một chỉ thị Upgrade(Q) ngay trước chỉ thị Write(Q), nếu chưa, nó xuất ra chỉ thị Lock-X(Q) ngay trước Write(Q). Tất cả các chốt giao dịch nhận được sẽ được tháo chốt sau khi giao dịch bàn giao hay bỏ dở.

GIAO THỨC DỰA TRÊN ĐỒ THỊ (Graph-Based Protocol)

Ta đã biết, trong trường hợp thiếu vắng các thông tin liên quan đến cách thức các hạng mục dữ liệu được truy xuất, giao thức chốt hai kỳ là cần và đủ để đảm bảo tính khả tuần tự. Nếu ta muốn phát triển các giao thức không là hai kỳ, ta cần các thông tin bổ xung trên cách thức mỗi giao dịch truy xuất CSDL. Có nhiều mô hình khác nhau về lượng thông tin được cung cấp. Mô hình đơn giản nhất đòi hỏi ta phải biết trước thứ tự trong đó các hạng mục dữ liệu sẽ được truy xuất. Với các thông tin như vậy, có thể xây dựng các giao thức chốt không là hai kỳ nhưng vẫn đảm bảo tính khả tuần tự xung đột.

Để có được hiểu biết trước như vậy, ta áp đặt một thứ tự bộ phận, ký hiệu , trên tập tất cả các hạng mục dữ liệu D ={ d1 , d2 , ..., dn }. Nếu di dj , bất kỳ giao dịch nào truy xuất cả di và dj phải truy xuất di trước khi truy xuất dj . Thứ tự bộ phận này cho phép xem D như một đồ thị định hướng phi chu trình, được gọi là đồ thị CSDL (DataBase Graph). Trong phần này, để đơn giản, ta hạn chế chỉ xét các đồ thị là các cây và ta sẽ đưa ra một giao thức đơn giản, được gọi là giao thức cây (tree protocol), giao thức này hạn chế chỉ dùng các chốt exclusive.

Trong giao thức cây, chỉ cho phép chỉ thị chốt Lock-X, mỗi giao dịch T có thể chốt một hạng mục dữ liệu nhiều nhất một lần và phải tuân theo các quy tắc sau:

1. Chốt đầu tiên bởi T có thể trên bất kỳ hạng mục dữ liệu nào
2. Sau đó, một hạng mục dữ liệu Q có thể bị chốt bởi T chỉ nếu cha của Q hiện đang bị chốt bởi T
3. Các hạng mục dữ liệu có thể được tháo chốt bất kỳ lúc nào

4. Một hạng mục dữ liệu đã bị chốt và được tháo chốt bởi T, không thể bị T chốt lại lần nữa.

Các lịch trình hợp lệ trong giao thức cây là khả tuần tự xung đột.

Ví dụ: Cây CSDL là:

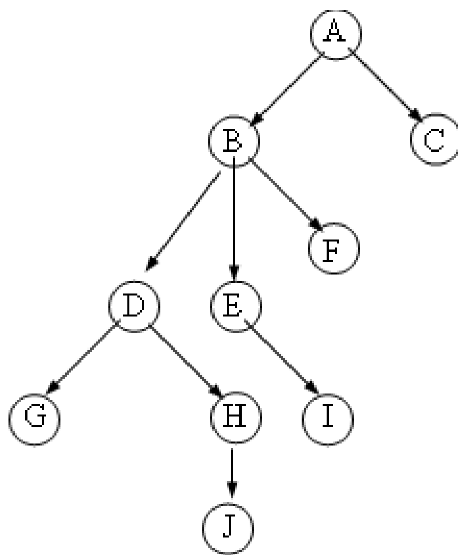


figure V-

Chỉ các chỉ thị chốt và tháo chốt của các giao dịch được trình bày:

T10 : Lock-X(B); Lock-X(E); Lock-X(D); Unlock(B); Unlock(E); Lock-X(G); Unlock(D); Unlock(G).

T11 : Lock-X(D); Lock-X(H); Unlock(D); Unlock(H).

T12 : Lock-X(B); Lock-X(E); Unlock(B); Unlock(E).

T13 : Lock-X(D); Lock-X(H); Unlock(D); Unlock(H).

figure V-

Một lịch trình tuân theo giao thức cây chứa tất cả bốn giao dịch trên được cho trong hình bên dưới. Ta nhận thấy, các lịch trình tuân thủ giao thức cây không chỉ là khả tuần tự xung đột mà còn đảm bảo không có dealock. Giao thức cây có mặt thuận lợi so với giao thức hai kỳ là tháo chốt có thể xảy ra sớm hơn. Việc tháo chốt sớm có thể dẫn đến rút ngắn thời gian chờ đợi và tăng tính cạnh tranh. Hơn nữa, do giao thức là không dealock, nên không có cuộn lại. Tuy nhiên giao thức cây có điểm bất lợi là, trong một vài trường hợp, một giao dịch có thể phải chốt những hạng mục dữ liệu mà nó không truy xuất. Chẳng hạn, một giao dịch cần truy xuất các hạng mục dữ liệu A và J trong đồ thị CSDL trên, phải chốt không chỉ A và J mà phải chốt cả các hạng mục B, D, H. Việc chốt bổ xung này có thể gây ra việc tăng tổng phí chốt, tăng thời gian chờ đợi và giảm tính cạnh tranh. Hơn nữa, nếu không biết trước các hạng mục dữ liệu nào sẽ cần thiết phải chốt, các giao dịch sẽ phải chốt gốc của cây mà điều này làm giảm mạnh tính cạnh tranh.

Đối với một tập các giao dịch, có thể có các lịch trình khả tuần tự xung đột không thể nhận được từ việc tuân theo giao thức cây. Có các lịch trình được sinh ra bởi tuân theo giao thức chốt hai kỳ nhưng không thể được sinh ra bởi tuân theo giao thức cây và ngược lại.

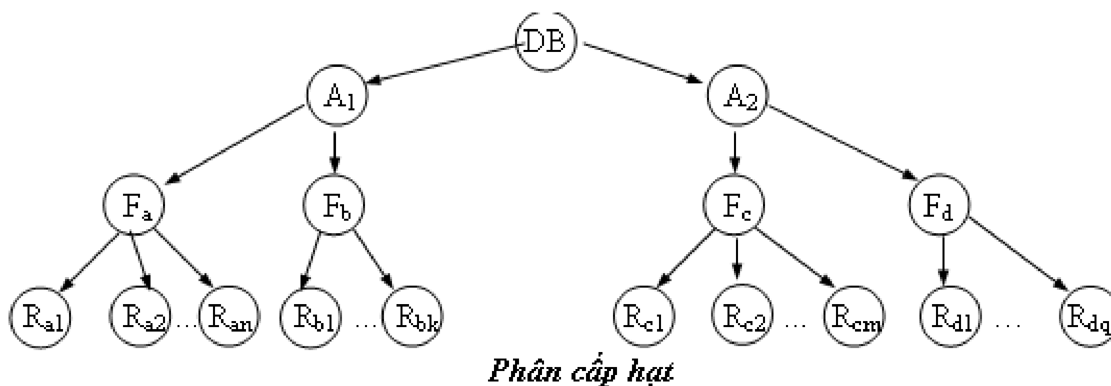
T10	T11	T12	T13
Lock-X(B)			
	Lock-X(D)		
	Lock-X(H)		
	Unlock(D)		
Lock-X(E)			

Lock-X(D)			
Unlock(B)			
Unlock(E)			
		Lock-X(B)	
		Lock-X(E)	
	Unlock(H)		
Lock-X(G)			
Unlock(D)			
			Lock-X(D)
			Lock-X(H)
			Unlock(D)
			Unlock(H)
		Unlock(E)	
		Unlock(B)	
Unlock(G)			
Lịch trình khả tuần tự dưới giao thức cây			

figure V-

ĐA HẠT (Multiple Granularity)

Trong các sơ đồ điều khiển cạnh tranh được mô tả trước đây, ta đã sử dụng hạng mục dữ liệu như đơn vị trên nó sự đồng bộ hoá được thực hiện. Tuy nhiên, có các hoàn cảnh trong đó việc nhóm một vài hạng mục dữ liệu và xử lý chúng như một đơn vị đồng bộ hoá mang lại nhiều lợi ích. Nừu một giao dich Ti phải truy xuất toàn bộ CSDL và giao thức chốt được sử dụng, khi đó Ti phải chốt mỗi hạng mục dữ liệu trong CSDL. Như vậy việc thực hiện các chốt này sẽ tiêu tốn một thời gian đáng kể. Sẽ hiệu quả hơn nếu Ti chỉ cần một yêu cầu chốt để chốt toàn bộ CSDL. Mặt khác, nếu Ti cần truy xuất chỉ một vài hạng mục dữ liệu, nó không cần thiết phải chốt toàn bộ CSDL vì như vậy sẽ giảm tính cạnh tranh. Như vậy, cái mà ta cần là một cơ chế cho phép hệ thống xác định nhiều mức hạt. Một cơ chế như vậy là cho phép các hạng mục dữ liệu có kích cỡ khác nhau và xác định một sự phân cấp các hạt dữ liệu, trong đó các hạt nhỏ được ẩn náu bên trong các hạt lớn. Sự phân cấp như vậy có thể được biểu diễn đồ thị như một cây. Một nút không là lá của cây đa hạt biểu diễn dữ liệu được kết hợp với con cháu của nó. Như một ví dụ minh hoạ, ta xét cây sau:



Phân cấp hạt

figure V-

Nó gồm bốn mức nút. Mức cao nhất là toàn bộ CSDL. Thấp hơn là các nút kiểu vùng: CSDL bao gồm các vùng này. Mỗi vùng lại có các nút kiểu file như các con của nó, mỗi vùng chứa đúng các file này và không file nào

nằm trong nhiều hơn một vùng. Cuối cùng, mỗi file có các nút con kiểu mẩu tin, không mẩu tin nào hiện diện trong hơn một file.

Mỗi nút trong cây có thể được chốt một các cá nhân. Như đã làm trong giao thức chốt hai kỳ, ta sẽ sử dụng các phương thức chốt shared và exclusive . Khi một giao dịch chốt một nút, trong phương thức shared hoặc exclusive, giao dịch cũng chốt tất cả các nút con cháu của nút này ở cùng phương thức. Ví dụ Ti chốt tường minh file Fb ở phương thức exclusive, nó đã chốt ẩn tất cả các mẩu tin của Fb cũng trong phương thức exclusive.

Giả sử giao dịch Tj muốn chốt mẩu tin Rb6 của file Fb. vì giao địch Ti đã chốt tường minh file Fb, mẩu tin Rb6 cũng bị chốt ẩn. Song làm thế nào để hệ thống biết được Tj có thể chốt Rb6 hay không: Tj phải duyệt cây từ gốc đến mẩu tin Rb6 , nếu có một nút bất kỳ tren đường dẫn bị chốt ở phương thức không tương thích, Tj phải chờ. Bây giờ, nếu Tk muốn chốt toàn bộ CSDL, nó phải chốt nút gốc. Tuy nhiên, do Ti hiện đang giữ một chốt trên Fb, một bộ phận của cây, nên Tk sẽ không thành công. Vậy làm thế nào để hệ có thể chốt được nút gốc: Một khả năng là tìm kiếm trên toàn bộ cây, giải pháp này phá huỷ hoàn toàn sơ đồ mục đích của sơ đồ chốt đa hạt. Một giải pháp hiệu quả hơn là đưa vào một lớp mới các phương thức chốt, được gọi là phương thức chốt tăng cường (intension lock mode). Nếu một nút bị chốt ở phương thức tăng cường, chốt tường minh được tiến hành ở mức thấp hơn của cây (hạt min hơn). Chốt tăng cường được được đặt trên tất cả các tổ tiên của một nút trước khi nút đó được chốt tường minh. Như vậy một giao dịch không cần thiết phải tìm kiếm toàn bộ cây để xác định nó có thể chốt một nút thành công hay không. Một giao dịch muốn chốt một nút, chẳng hạn Q, phải duyệt một đường dẫn từ gốc đến Q, trong khi duyệt cây, giao dịch chốt các nút trên đường đi ở phương thức tăng cường.

Có một phương thức tăng cường kết hợp với phương thức shared và một với phương thức exclusive. Nếu một nút bị chốt ở phương thức tăng cường shared (IS), chốt tường minh được tiến hành ở mức thấp hơn trong cây, nhưng chỉ là một các chốt phương thức shared. Tương tự, nếu một nút bị chốt ở phương thức tăng cường exclusive (IX), chốt tường

minh được tiến hành ở mức thấp hơn với các chốt exclusive hoặc shared. Nếu một nút bị chốt ở phương thức shared và phương thức tăng cường exclusive (SIX), cây con có gốc là nút này bị chốt tường minh ở phương thức shared và chốt tường minh được tiến hành ở mức thấp hơn với các chốt exclusive. Hàm tính tương thích đối với các phương thức chốt này được cho bởi ma trận:

	IS	IX	S	SIX	X
IS	True	True	True	True	False
IX	True	True	False	False	False
S	True	False	True	False	False
SIX	True	False	False	False	False
X	False	False	False	False	False

figure V-

Giao thức chốt đa hạt dưới đây đảm bảo tính khả tuần tự. Mỗi giao địch T có thể chốt một nút Q theo các quy tắc sau:

1. Hàm tương thích chốt phải được kiểm chứng
2. Gốc của cây phải được chốt đầu tiên, và có thể được chốt ở bất kỳ phương thức nào
3. Một nút Q có thể được chốt bởi T ở phương thức S hoặc IS chỉ nếu cha của Q hiện đang bị chốt bởi T ở hoặc phương thức IX hoặc phương thức IS.

4. Một nút Q có thể được chốt bởi T ở phương thức X, SIX hoặc IX chỉ nếu cha của Q hiện đang bị chốt ở hoặc phương thức IX hoặc phương thức SIX
5. T có thể chốt một nút chỉ nếu trước đó nó chưa tháo chốt một nút nào.
6. T có thể tháo chốt một nút Q chỉ nếu không con nào của Q hiện đang bị chốt bởi T

Ta thấy rằng giao thức đa hạt yêu cầu các chốt được tậu theo thứ tự Top-Down, được tháo theo thứ tự Bottom-Up.

Ví dụ: Xét cây phân cấp hạt như trên và các giao dịch sau:

- Giả sử giao dịch T18 đọc mẩu tin Ra2 của file Fa. Khi đó T18 cần phải chốt DB, vùng A1 và Fa ở phương thức IS và Ra2 ở phương thức S: Lock-IS(DB); Lock-IS(A1); Lock-IS(Fa); lock-S(Ra2)
- Giả sử giao dịch T19 sửa đổi mẩu tin Ra9 trong file Fa , khi đó T19 cần phải chốt CSDL, vùng A1 và file Fa ở phương thức IX và Ra9 ở phương thức X: Lock-IX(DB); Lock-IX(A1); lock-IX(Fa); lock-X(Ra9)
- Giả sử giao dịch T20 đọc tất cả các mẩu tin của file Fa , khi đó T20 cần phải chốt CSDL, và vùng A1 ở phương thức IS và chốt Fa ở phương thức S: Lock-IS(DB); Lock-IS(A1); Lock-S(Fa)
- Giả sử giao dịch T21 đọc toàn bộ CSDL, nó có thể làm điều đó sau khi chốt CSDL ở phương thức S: Lock-S(DB)

Chú ý rằng T18, T20 và T21 có thể truy xuất đồng thời CSDL, giao dịch T19 có thể thực hiện cạnh tranh với T18 nhưng không với T20 hoặc T21

GIAO THỨC DỰA TRÊN TEM THỜI GIAN (Timestamp-based protocol)

TEM THỜI GIAN (Timestamp)

Ta kết hợp với mỗi giao dịch Ti trong hệ thống một tem thời gian cố định duy nhất, được biểu thị bởi TS(Ti). Tem thời gian này được gán bởi hệ

CSDL trước khi giao dịch Ti bắt đầu thực hiện. Nếu một giao dịch Ti đã được gán tem thời gian TS(Ti) và một giao dịch mới Tj đi vào hệ thống, khi đó TS(Ti) < TS(Tj). Có hai phương pháp đơn giản để thực hiện sơ đồ này:

1. Sử dụng giá trị của đồng hồ hệ thống như tem thời gian: Một tem thời gian của một giao dịch bằng giá trị của đồng hồ khi giao dịch đi vào hệ thống.
2. Sử dụng bộ đếm logic: bộ đếm được tăng lên mỗi khi một tem thời gian đã được gán, tem thời gian của một giao dịch bằng với giá trị của bộ đếm khi giao dich đi vào hệ thống.

Tem thời gian của các giao dịch xác định thứ tự khả tuần tự. Như vậy, nếu TS(Ti) < TS(Tj), hệ thống phải đảm bảo rằng lịch trình được sinh ra là tương đương với một lịch trình tuần tự trong đó Ti xuất hiện trước Tj .

Để thực hiện sơ đồ này, ta kết hợp với mỗi hạng mục dữ liệu Q hai giá trị tem thời gian:

- W-timestamp(Q) biểu thị tem thời gian lớn nhất của giao dịch bất kỳ đã thực hiện Write(Q) thành công
- R-timestamp(Q) biểu thị tem thời gian lớn nhất của giao dịch bất kỳ đã thực hiện Read(Q) thành công

Các tem thời gian này được cập nhật mỗi khi một Write hoặc một Read mới được thực hiện.

GIAO THỨC THỨ TỰ TEM THỜI GIAN (Timestamp-Ordering Protocol)

Giao thức thứ tự tem thời gian đảm bảo rằng các Write và Read xung đột bất kỳ được thực hiện theo thứ tự tem thời gian. Giao thức này hoạt động như sau:

1. Giả sử giao dịch Ti phát ra Read(Q).

- Nếu TS(Ti) < W-Timestamp(Q), Ti cần đọc một giá trị của Q đã được viết rồi. Do đó, hoạt động Read bị vứt bỏ và Ti bị cuộn lại.
- Nếu TS(Ti)   W-Timestamp(Q), hoạt động Read được thực hiện và R-Timestamp được đặt bằng giá trị lớn nhất trong hai giá trị R-Timestamp và TS(Ti).

Giả sử giao dịch Ti phát ra Write(Q).

- Nếu TS(Ti) < R-Timestamp(Q), Giá trị của Q mà Ti đang sinh ra được giả thiết là để được dùng cho các giao dịch đi sau nó (theo trình tự thời gian), nhưng nay không cần đến nũa. Do vậy, hoạt động Write này bị vứt bỏ và Ti bị cuộn lại
- Nếu TS(Ti) < W-Timestamp(Q), Ti đang thử viết một giá trị đã quá hạn của Q, Từ đó, hoạt động Write bị vứt bỏ và Ti bị cuộn lại
- Ngoài ra, hoạt động Write được thực hiện và W-Timestamp(Q) được đặt là TS(Ti)

Một giao dịch Ti bị cuộn lại bởi sơ đồ điều khiển cạnh tranh như kết quả của hoạt động Read hoặc Write đang được phát ra, được gán với một tem thời gian mới và được tái khởi động lại (được xem như một giao dịch mới tham gia vào hệ thống)

Ta xét các giao dịch T14 và T15 được xác định như dưới đây:

T14 :Read(B);

Read(A);

Display(A+B);.

T15 :Read(B);

B:=B-50;

Write(B);

Read(A);

A:=A+50;

Write(A);

Display(A+B).

figure V-

Ta giả thiết rằng một giao dịch được gán cho một tem thời gian ngay trước chỉ thị đầu tiên của nó. Như vậy, lịch trình schedule-3 dưới đây có TS(T14) < TS(T15), và là một lịch trình hợp lệ dưới giao thức tem thời gian:

T_{14}	T_{15}
Read(B)	
	Read(B)
	B:=B-50
	Write(B)
Read(A)	
	Read(A)
Display(A+B)	
	A:=A+50
	Write(A)
	Display(A+B)

Schedule-3

figure V- 1

Giao thức thứ tự tem thời gian đảm bảo tính khả tuần tự xung đột và không dealock.

QUY TẮC VIẾT THOMAS (Thomas' Write rule)

Một biến thể của giao thức tem thời gian cho phép tính cạnh tranh cao hơn giao thức thứ tự tem thời gian. Trước hết ta xét lịch trình schedule-4 sau:

T16	T17
Read(Q)	
	Write(Q)
Write(Q)	
Schedule-4	

figure V-

Nếu áp dụng giao thức thứ tự tem thời gian, ta có TS(T16) < TS(T17). Hoạt dộng Read(Q) của T16 và Write(Q) của T17 thành công, khi T16 toan thực hiện hoạt động Write(Q) của nó, vì TS(T16) < TS(T17) = W-timestamp(Q), nên Write(Q) của T16 bị vứt bỏ và giao dịch T16 bị cuộn lại. Sự cuộn lại này là không cần thiết. Nhận xét này cho ta một sửa đổi phiên bản giao thức thứ tự tem thời gian:

Các quy tắc giao thức đối với Read không thay đổi, các quy tắc đối với Write được thay đổi chút ít như sau:

Giả sử giao dịch Ti phát ra Write(Q).

1. Nếu TS(Ti) < R-timestamp(Q), Giá trị của Q mà Ti đang sinh ra được giả thiết là để được dùng cho các giao dịch đi sau nó (theo trình tự thời gian), nhưng nay không cần đến nữa. Do vậy, hoạt động Write này bị vứt bỏ và Ti bị cuộn lại.
2. Nếu TS(Ti) < W-timestamp(Q), Ti đang thử viết một giá trị lỗi thời của Q. Do vậy, hoạt động Write này có thể bị bỏ lơ (không được thực hiện, nhưng Ti không bị cuộn lại).
3. Ngoài ra, hoạt động Write được thực hiện và W-timestamp(Q) được đặt là TS(Ti).

Sự sửa đổi đối với giao thức thứ tự tem thời gian này được gọi là quy tắc viết Thomas. Quy tắc viết Thomas cho khả năng sinh các lịch trình khả

tuần tự mà các giao thức thức trước đây không thể.

GIAO THỨC DỰA TRÊN TÍNH HỢP LỆ

Trong trường hợp đa số các giao dịch trong hệ thống là các giao dịch chỉ đọc (read-only), tỷ suất xung đột giữa các giao dịch là thấp. Như vậy nhiều giao dịch trong chúng thực hiện thiếu sự giám sát của sơ đồ điều khiển cạnh tranh cũng vẫn giữ cho hệ thống ở trạng thái nhất quán. Hơn nữa, một sơ đồ điều khiển cạnh tranh đưa vào một tổng phí đáng kể (cho thực hiện mã lệnh, thời gian chờ của giao dịch ...). Việc tìm một sơ đổ với tổng phí nhỏ là một mục tiêu. Nhưng khó khăn là ta phải biết trước những giao dịch sẽ bị dính líu vào một xung đột. Để có được các hiểu bét đó, ta cần một sơ đồ để giám sát hệ thống.

Ta giả thiết rằng mỗi giao dịch Ti , trong thời gian sống của nó, thực hiện trong hai hoặc ba kỳ khác nhau, phụ thuộc vào nó là một giao dịch chỉ đọc hay là một giao dịch cập nhật. Các kỳ này, theo thứ tự, là như sau:

1. Kỳ đọc. Trong kỳ này, các giá trị của các hạng mục dữ liệu khác nhau được đọc vào các biến cục bộ của Ti . Tất cả các hoạt động Write được thực hiện trên các biến cục bộ tạm, không cập nhật CSDL hiện hành.
2. Kỳ hợp lệ. Giao dịch Ti thực hiện một phép kiểm thử sự hợp lệ để xác định xem nó có thể sao chép đến CSDL các biến cục bộ tạm chứa các kết quả của các hoạt Write mà không vi phạm tính khả tuần tự xung đột hay không.
3. Kỳ viết. Nếu Ti thành công trong kỳ hợp lệ, các cập nhật hiện hành được áp dụng vào CSDL, nếu không Ti bị cuộn lại.

Mỗi giao dịch phải trải qua ba kỳ theo thứ tự trên, tuy nhiên, ba kỳ của các giao dịch đang thực hiện cạnh tranh có thể đan xen nhau.

Các kỳ đọc và kỳ viết tự nó đã rõ ràng. Chỉ có kỳ hợp lệ là cần thảo luận thêm. Để thực hiện kiểm thử sự hợp lệ, ta cần biết khi nào các kỳ khác

nhau của giao dịch Ti xảy ra. Do vậy, ta sẽ kết hợp ba tem thời gian với giao dịch Ti :

1. Start(Ti). Thời gian khi Ti bắt đầu sự thực hiện.
2. Validation(Ti). Thời gian khi Ti kết thúc kỳ đọc và khởi động kỳ hợp lệ.
3. Finish(Ti). Thời gian khi Ti kết thúc kỳ viết.

Ta xác định thứ tự khả tuần tự bởi kỹ thuật thứ tự tem thời gian sử dụng giá trị tem thời gian Validation(Ti). Như vậy, giá trị TS(Ti) = Validation(Ti) và nếu TS(Tj) < TS(Tk) thì bất kỳ lịch trình nào được sinh ra phải tương đương với một lịch trình tuần tự trong đó giao dịch Ti xuất hiện trước giao dịch Tk . Lý dota chọn Validation(Ti) như tem thời gian của Ti , mà không chọn Start(Ti), là vì ta hy vọng thời gian trả lời sẽ nhanh hơn.

Phép kiểm thử hợp lệ đối với Tj đòi hỏi rằng, đối với mỗi giao dịch Ti với TS(Ti) < TS(Tj), một trong các điều kiện sau phải được thoả mãn:

1. Finish(Ti) < Start(Tj). Do Ti hoàn tất sự thực hiện của nó trước khi Tj bắt đầu, thứ tự khả tuần tự được duy trì.
2. Tập các hạng mục dữ liệu được viết bởi Ti không giao với tập các hạng mục dữ liệu được đọc bởi Tj và Ti hoàn tất kỳ viết của nó trước khi Tj bắt đầu kỳ hợp lệ (Start(Tj) < Finish(Ti) < Validation(Tj)). Điều kiện này đảm bảo rằng các viết của Ti và Tj là không chồng chéo. Do các viết của Ti không ảnh hưởng tới đọc của Tj và do Tj không thể ảnh hưởng tới đọc của Ti , thứ tự khả tuần tự được duy trì.

Lịch trình schedule-5 cho ta một minh hoạ về giao thức dựa trên tính hợp lệ:

T14	T15

Read(B)	
	Read(B)
	B:=B-50
	Read(A)
	A:=A+50
Read(A)	
Xác nhận tính hợp lệ	
Display(A+B)	
	Xác nhận tính hợp lệ
	Write(B)
	Write(A)

figure V-

Sơ đồ hợp lệ tự động canh chừng việc cuộn lại hàng loạt, do các Write hiện tại xảy ra chỉ sau khi giao dịch phát ra Write đã bàn giao.

CÁC SƠ ĐỒ ĐA PHIÊN BẢN (Multiversion Schemes)

Các sơ đồ điều khiển cạnh tranh được thảo luận trước đây đảm bảo tính khả tuần tự hoặc bởi làm trễ một hoạt động hoặc bỏ dở giao dịc đã phát ra hoạt động đó. Chẳng hạn, một hoạt động Read có thể bị làm trễ vì giá trị thích hợp còn chưa được viết hoặc nó có thể bị vứt bỏ vì giá trị mà nó muốn đọc đã bị viết đè rồi. Các khó khăn này có thể được che đi nếu bản sao cũ của mỗi hạng mục dữ liệu được giữ trong một hệ thống.

Trong các hệ CSDL đa phiên bản, mỗi hoạt động Write(Q) tạo ra một bản mới của Q. Khi một hoạt động Read(Q) được phát ra, hệ thống chọn lựa một trong các phiên bản của Q để đọc. Sơ đồ điều khiển cạnh tranh phải đảm bảo rằng việc chọn lựa này được tiến hành sao cho tính khả tuần tự được đảm bảo. Do lý do hiệu năng, một giao dịch phải có khả năng xác định dễ dàng và mau chóng phiên bản dhạng mục dữ liệu sẽ đọc.

THỨ TỰ TEM THỜI GIAN ĐA PHIÊN BẢN

Kỹ thuật chung được dùng trong các sơ đồ đa phiên bản là tem thời gian. Ta kết hợp với một giao dịch một tem thời gian tĩnh duy nhất, ký hiệu TS(Ti). Tem thời gian này được gán trước khi khi giao dịch bắt đầu sự thực hiện. Mỗi hạng mục dữ liệu Q kết hợp với một dãy <Q1, Q2, ... , Qm> mỗi phiên bản Qk chứa ba trường dữ liệu:

- Content là giá trị của phiên bản Qi
- W-timestamp(Qk) là tem thời gian của giao dịch đã tạo ra phiên bản Qk
- R-timestamp(Qk) là tem thời gian lớn nhất của giao dịch đã đọc thành công phiên bản Qk

Một giao dịch, gọi là Ti , tạo ra phiên bản Qk của hạng mục dữ liệu Q bằng cách phát ra một hoạt động Write(Q). Trường Content của phiên bản chứa giá trị được viết bởi Ti . W-timestamp và R-timestamp được khởi động là TS(Ti). Giá trị R-timestamp được cập nhật mỗi khi một giao dịch Tj đọc nội dung của Qk và R-timestamp(Qk) < TS(Tj).

Sơ đồ tem thời gian đa phiên bản dưới đây sẽ đảm bảo tính khả tuần tự. Sơ đồ hoạt động như sau: giả sử Tj phát ra một hoạt động Read(Q) hoặc Write(Q). Qk ký hiệu phiên bản của Q, tem thời gian viết của nó là tem thời gian viết lớn nhất nhỏ hơn hoặc bằng TS(Tj).

1. Nếu giao dịch Tj phát ra một Read(Q), khi đó giá trị trả lại là nội dung của phiên bản Qk

2. Nếu Tj phát ra một Write(Q) và nếu TS(Tj) < R-timestamp(Qk) khi đó giao dịch Tj bị cuộn lại. Nếu không, nếu TS(Tj) = W-timestamp(Qk) nội dung của Qk bị viết đè, khác đi một phiên bản mới của Q được tạo.

Các phiên bản không còn được dùng đến nữa bị xoá đi dựa trên quy tắc sau: Giả sử có hai phiên bản Qi và Qj của một hạng mục dữ liệu và cả hai phiên bản này cùng có W-timestamp nhỏ hơn tem thời gian của giao dịch già nhất trong hệ thống, khi đó phiên bản già hơn trong hai phiên bản Qi và Qj sẽ không còn được dùng nữa và bị xoá đi.

Sơ đồ thứ tự tem thời gian đa phiên bản có tính chất hay đó là một yêu cầu Read không bao giờ thất bại và không phải chờ đợi. Trong một hệ thống mà hoạt động Read xảy ra nhiều hơn Write cái lợi này là đáng kể. Tuy nhiên có vài điều bất lợi của sơ đồ này là: thứ nhất đọc một hạng mục dữ liệu cũng đòi hỏi cập nhật trường R-timestamp, thứ hai là xung đột giữa các giao dịch được giải quyết bằng cuộn lại.

CHỐT HAI KỲ ĐA PHIÊN BẢN

Giao thức chốt hai kỳ đa phiên bản cố gắng tổ hợp những ưu điểm của điều khiển cạnh tranh với các ưu điểm của chốt hai kỳ. Giao thức này phân biệt các giao dịch chỉ đọc và các giao dịch cập nhật.

Các giao dịch cập nhật thực hiện chốt hai kỳ nghiêm khắc (các chốt được giữ đến tận khi kết thúc giao dịch). Mỗi hạng mục dữ liệu có một tem thời gian. Tem thời gian trong trường hợp này không là tem thời gian dựa trên đồng hồ thực mà là một bộ đếm, sẽ được gọi là TS-counter.

Các giao dịch chỉ đọc được gán tem thời gian là giá trị hiện hành của TS-counter trước khi chúng bắt đầu sự thực hiện: chúng tuân theo giao thức thứ tự tem thời gian đa phiên bản để thực hiện đọc. Như vậy, khi một giao dịch chỉ đọc Ti phát ra một Read(Q), giá trị trả lại là nội dung của phiên bản mà tem thời gian của nó là tem thời gian lớn nhất nhỏ hơn TS(Ti).

Khi một giao dịch cập nhật đọc một hạng mục, nó tậu một chốt shared trên hạng mục, rồi đọc phiên bản mới nhất của hạng mục (đối với nó). Khi một giao dịch cập nhật muốn viết một hạng mục, đầu tiên nó tậu một chốt exclusive trên hạng mục này, rồi tạo ra một phiên bản mới cho hạng mục. Write được thực hiện trên phiên bản mới này và tem thời gian của phiên bản mới được khởi động là + .

Khi một giao dịch cập nhật Ti hoàn tất các hoạt động của nó, nó thực hiện xử lý bàn giao như sau: Đầu tiên, Ti đặt tem thời gian trên mỗi phiên bản nó đã tạo là TS-counter+1; sau đó Ti tăng TS-counter lên 1. Chỉ một giao dịch cập nhật được phép thực hiện xử lý bàn giao tại một thời điểm.

Các phiên bản bị xoá cùng kiểu cách với thứ tự tem thời gian đa phiên bản.

QUẢN TRỊ DEADLOCK

Một hệ thống ở trạnh thái deadlock nếu tồn tại một tập hợp các giao dịch sao cho mỗi giao dịch trong tập hợp đang chờ một giao dịch khác trong tạp hợp. Chính xác hơn, tồn tại một tập các giao dịch { T0 , T2 , ..., Tn } sao cho T0 đang chờ một hạng mục dữ liệu được giữ bởi T1 , T1 đang chờ một hạng mục dữ liệu đang bị chiếm bởi T2 , ..., Tn-1 đang chờ một hạng mục dữ liệu được giữ bởi Tn và Tn đang chờ một hạng mục T0 đang chiếm. Không một giao dịch nào có thể tiến triển được trong tình huống như vậy. Một cách chữa trị là viện dẫn một hành động tẩy rửa, chẳng hạn cuộn lại một vài giao dịch tham gia vào deadlock.

Có hai phương pháp chính giải quyết vấn đề deadlock: Ngăn ngừa deadlock, phát hiện deadlock và khôi phục. Giao thức ngăn ngừa deadlock đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Sơ đồ phát hiện deadlock và khôi phục (deadlock-detection and deadlock-recovery scheme) cho phép hệ thống đi vào trạng thái deadlock và sau đó cố gắng khôi phục. Cả hai phương pháp đều có thể dẫn đến việc cuộn lại giao dịch. Phòng ngừa deadlock thường được sử dụng nếu xác suất hệ thống đi vào deadlock cao, phát hiện và khôi phục hiệu quả hơn trong các trường hợp còn lại.

PHÒNG NGỪA DEADLOCK (Deadlock prevention)

Có hai cách tiếp cận phòng ngừa deadlock: Một đảm bảo không có chờ đợi vòng tròn xảy ra bằng cách sắp thứ tự các yêu cầu chốt hoặc đòi hỏi tất cả các chốt được tậu cùng nhau. Một cách tiếp cận khác gần hơn với khắc phục deadlock và thực hiện cuộn lại thay vì chờ đợi một chốt. Chờ đợi là tiểm ẩn của deadlock.

Sơ đồ đơn giản nhất dưới cách tiếp cận thứ nhất đòi hỏi mỗi giao dịch chốt tất cả các hạng mục dữ liệu trước khi nó bắt đầu thực hiện. Hơn nữa, hoặc tất cả được chốt trong một bước hoặc không hạng mục nào được chốt. Giao thức này có hai bất lợi chính: một là khó dự đoán, trước khi giao dịch bắt đầu, các hạng mục dữ liệu nào cần được chốt, hai là hiệu suất sử dụng hạng mục dữ liệu rất thấp do nhiều hạng mục có thể bị chốt nhưng không được sử dụng trong một thời gian dài.

Sơ đồ phòng ngừa deadlock khác là áp đặt một thứ tự bộ phận trên tất cả các hạng mục dữ liệu và yêu cầu một giao dịch chốt một hạng mục dữ liệu theo thứ tự được xác định bởi thứ tự bộ phận này.

Cách tiếp cận thứ hai để phòng ngừa deadlock là sử dụng ưu tiên và cuộn lại quá trình. Với ưu tiên, một giao dịch T2 yêu cầu một chốt bị giữ bởi giao dịch T1 , chốt đã cấp cho T1 có thể bị lấy lại và cấp chgo T2 , T1 bị cuộn lại. Để điều khiển ưu tiên, ta gán một tem thời gian duy nhất cho mỗi giao dịch. Hệ thống sử dụng các tem thời gian này để quyết định một giao dịch phải chờ hay cuộn lại. Việc chốt vẫn được sử dụng để điều khiển cạnh tranh. Nếu một giao dịch bị cuộn lại, nó vẫn giữ tem thời gian cũ của nó khi tái khởi động. Hai sơ đồ phòng ngừa deadlock sử dụng tem thời gian khác nhau được đề nghị:

1. Sơ đồ Wait-Die dựa trên kỹ thuật không ưu tiên. Khi giao dịch Ti yêu cầu một hạng mục dữ liệu bị chiếm bởi Tj , Ti được phép chờ chỉ nếu nó có tem thời gian nhỏ hơn của Tj nếu không Ti bị cuộn lại (die).
2. Sơ đồ Wound-Wait dựa trên kỹ thuật ưu tiên. Khi giao dịch Ti yêu cầu một hạng mục dữ liệu hiện đang bị giữ bởi Tj , Ti được phép

chờ chỉ nếu nó có tem thời gian lớn hơn của Tj , nếu không Tj bị cuộn lại (Wounded).

Một điều quan trọng là phải đảm bảo rằng, mỗi khi giao dịch bị cuộn lại, nó không bị chết đói (starvation) có nghĩa là nó sẽ không bị cuộn lại lần nữa và được phép tiến triển.

Cả hai sơ đồ Wound-Wait và Wait-Die đều tránh được sự chết đói: tại một thời điểm, có một giao dịch với tem thời gian nhỏ nhất. Giao dịch này không thể bị yêu cầu cuộn lại trong cả hai sơ đồ. Do tem thời gian luôn tăng và do các giao dịch không được gán tem thời gian mới khi chúng bị cuộn lại, một giao dịch bị cuộn lại sẽ có tem thời gian nhỏ nhất (vào thời gian sau) và sẽ không bị cuộn lại lần nữa.

Tuy nhiên, có những khác nhau lớn trong cách thức hoạt động của hai sơ đồ:

- Trong sơ đồ Wait-Die, một giao dịch già hơn phải chờ một giao dịch trẻ hơn giải phóng hạng mục dữ liệu. Như vậy, giao dịch già hơn có xu hướng bị chờ nhiều hơn. Ngược lại, trong sơ đồ Wound-Wait, một giao dịch già hơn không bao giờ phải chờ một giao dịch trẻ hơn.
- Trong sơ đồ Wait-Die, nếu một giao dịch Ti chết và bị cuộn lại vì nó đòi hỏi một hạng mục dữ liệu bị giữ bởi giao dịch Tj , khi đó Ti có thể phải tái phát ra cùng dãy các yêu cầu khi nó khởi động lại. Nếu hạng mục dữ liệu vẫn bị chiếm bởi Tj , Ti bị chết lần nữa. Như vậy, Ti có thể bị chết vài lần trước khi tậu được hạng mục dữ liệu cần thiết. Trong sơ đồ Wound-Wait, Giao dịch Ti bị thương và bị cuộn lại do Tj yêu cầu hạng mục dữ liệu nó chiếm giữ. Khi Ti khởi động lại, và yêu cầu hạng mục dữ liệu, bây giờ, đang bị Tj giữ, Ti chờ. Như vậy, có ít cuộn lại hơn trong sơ đồ Wound-Wait.

Một vấn đề nổi trội đối với cả hai sơ đồ là có những cuộn lại không cần thiết vẫn xảy ra.

SƠ ĐỒ DỰA TRÊN TIMEOUT

Một cách tiếp cận khác để quản lý deadlock được dựa trên lock timeout. Trong cách tiếp cận này, một giao dịch đã yêu cầu một chốt phải chờ nhiều nhất một khoảng thời gian xác định. Nếu chốt không được cấp trong khoảng thời gian này, giao dịch được gọi là mãn kỳ (time out), giao dịch tự cuộn lại và khởi động lại. Nếu có một deadlock, một hoặc một vài giao dịch dính líu đến deadlock sẽ time out và cuộn lại, để các giao dịch khác tiến triển. Sơ đồ này nằm trung gian giữa phòng ngừa deadlock và phát hiện và khôi phục deadlock.

Sơ đồ timeout dễ thực thi và hoạt động tốt nếu giao dịch ngắn và nếu sự chờ đợi lâu là do deadlock. Tuy nhiên, khó quyết định được khoảng thời gian timeout. Sơ đồ này cũng có thể đưa đến sự chết đói.

PHÁT HIỆN DEADLOCK VÀ KHÔI PHỤC

Nếu một hệ thống không dùng giao thức phòng ngừa deadlock, khi đó sơ đồ phát hiện và khôi phục phải được sử dụng. Một giải thuật kiểm tra trạng thái của hệ thống được gọi theo một chu kỳ để xác định xem deadlock có xẩy ra hay không. Nếu có, hệ thống phải khôi phục lại từ deadlock, muốn vậy hệ thống phải:

- Duy trì thông tin về sự cấp phát hiện hành các hạng mục dữ liệu cho các giao dịch cũng như các yêu cầu hạng mục dữ liệu chưa được chưa được giải quyết.
- Cung cấp một thuật toán sử dụng các thông tin này để xác định hệ thống đã đi vào trạng thái deadlock chưa.
- Phục hồi từ deadlock khi phát hiện được deadlock đã xảy ra.

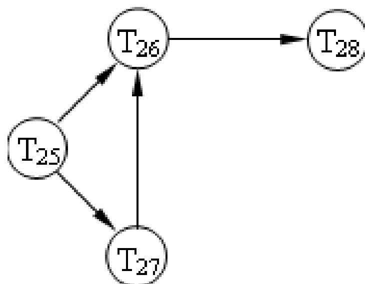
PHÁT HIỆN DEADLOCK

Deadlock có thể mô tả chính xác bằng đồ thị định hướng được gọi là đồ thị chờ (wait for graph). Đồ thị này gồm một cặp G = < V, E >, trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh gồm tất cả các giao

dịch trong hệ thống. Mỗi phần tử của E là một cặp Ti   Tj , nó chỉ ra rằng Ti chờ Tj giải phóng một hạng mục dữ liệu nó cần.

Khi giao dịch Ti yêu cầu một hạng mục dữ liệu đang bị giữ bởi giao dịch Tj khi đó cung Ti   Tj được xen vào đồ thị. Cạnh này bị xoá đi chỉ khi giao dịch Tj không còn giữ hạng mục dữ liệu nào mà Ti cần.

Deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa một chu trình. Mỗi giao dịch tham gia vào chu trình này được gọi là bị deadlock. Để phát hiện deadlock, hệ thống phải duy trì đồ thị chờ và gọi theo một chu kỳ thủ tục tìm kiếm chu trình. Ta xét ví dụ sau:



Đồ thị chờ (phi chu trình)

figure V-

Do đồ thị không có chu trình nên hệ thông không trong trạng thái deadlock. Bây giờ, giả sử T28 yêu cầu một hạng mục dữ liệu được giữ bởi T27 , cung T28   T27 được xen vào đồ thị, điều này dẫn đến tồn tại một chu trình T26   T27   T28   T26 có nghĩa là hệ thống rơi vào tình trạng deadlock và T26 , T27 , T28 bị deadlock.

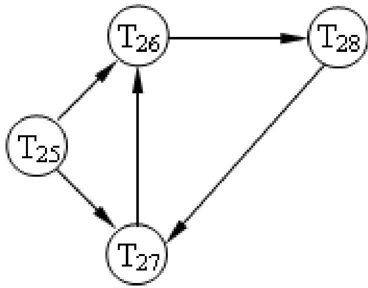


figure V-

Vấn đề đặt ra là khi nào thì chạy thủ tục phát hiện ? câu trả lời phụ thuộc hai yêu tố sau:

1. Deadlock thường xảy ra hay không ?
2. Bao nhiêu giao dịch sẽ bị ảnh hưởng bởi deadlock

Nếu deadlock thường xảy ra, việc chạy thủ tục phát hiện diễn ra thường xuyên hơn. Các hạng mục dữ liệu được cấp cho các giao dịch bị deadlock sẽ không sẵn dùng cho các giao dịch khác đến khi deadlock bị phá vỡ. Hơn nữa, số chu trình trong đồ thị có thể tăng lên. Trong trường hợp xấu nhất, ta phải gọi thủ tục phát hiện mỗi khi có một yêu cầu cấp phát không được cấp ngay.

KHÔI PHỤC TỪ DEADLOCK

Khi thuật toán phát hiện xác định được sự tồn tại của deadlock, hệ thống phải khôi phục từ deadlock. Giải pháp chung nhất là cuộn lại một vài giao dịch để phá vỡ deadlock. Ba việc cần phải làm là:

1. Chọn nạn nhân. Đã cho một tập các giao dịch bị deadlock, ta phải xác định giao dịch nào phải cuộn lại để phá vỡ deadlock. Ta sẽ cuộn lại các giao dịch sao cho giá phải trả là tối thiểu. Nhiều nhân tố xác định giá của cuộn lại:

1. Giao dịch đã tính toán được bao lâu và bao lâu nữa.
2. Giao dịch đã sử dụng bao nhiêu hạng mục dữ liệu

3. Giao dịch cần bao nhiêu hạng mục dữ liệu nữa để hoàn tất.
4. Bao nhiêu giao dịch bị cuộn lại.
5. Cuộn lại (Rollback). Mỗi khi ta đã quyết định được giao dịch nào phải bị cuộn lại, ta phải xác định giao dịch này bị cuộn lại bao xa. Giải pháp đơn giản nhất là cuộn lại toàn bộ: bỏ dở giao dịch và bắt đầu lại nó. Tuy nhiên, sẽ là hiệu quả hơn nếu chỉ cuộn lại giao dịch đủ xa như cần thiết để phá vỡ deadlock. Nhưng phương pháp này đòi hỏi hệ thống phải duy trì các thông tin bổ xung về trạng thái của tất cả các giao dịch đang chạy.
6. Sự chết đói (Starvation). Trong một hệ thống trong đó việc chọn nạn nhân dựa trên các nhân tố giá, có thể xảy ra là một giao dịch luôn là nạn nhân của việc chọn này và kết quả là giao dịch này không bao giờ có thể hoàn thành. Tình huống này được gọi là sự chết đói. Phải đảm bảo việc chọn nạn nhân không đưa đến chết đói. Một giải pháp xem số lần bị cuộn lại của một giao dịch như một nhân tố về giá.

BÀI TẬP CHƯƠNG V

V.1Chứng tỏ rằng giao thức chốt hai kỳ đảm bảo tính khả tuần tự xung đột và các giao dịch có thể được làm tuần tự tương ứng với các điểm chốt của chúng.

V.2 Xét hai giao dịch sau:

T31 :Read(A);

Read(B);

If A=0 then B:=B+1;

Write(B);

T32 :Read(B);

Read(A);

If B=0 then A:=A+1;

Write(A);

Thêm các chỉ thị chốt và tháo chốt vào hai giao dịch T31 và T32 sao cho chúng tuân theo giao thức chốt hai kỳ. Sự thực hiện các giao dịch này có thể dẫn đến deadlock ?

V.3Nêu các ưu điểm và nhược điểm của giao thức chốt hai kỳ nghiêm ngặt.

V.4Nêu các ưu điểm và nhược điểm của giao thức chốt hai kỳ nghiêm khắc.

V.5Bộ quản trị điều khiển cạnh tranh của một hệ CSDL phải quyết định cấp cho một đòi hỏi chốt hoặc bắt giao dịch yêu cầu phải chờ. Hãy thiết kế một cấu trúc dữ liệu (tiết kiệm không gian) cho phép bộ quản trị điều khiển cạnh tranh cho ra quyết định mau chóng.

V.6Xét sự mở rộng thành giao thức cây-chốt sau. Nó cho phép cả chốt shared và exclusive:

1. Một giao dịch chỉ đọc chỉ có thể yêu cầu các chốt shared. Một giao dịch cập nhật chỉ có thể yêu cầu các chốt exclusive
2. Mỗi giao dịch phải tuân theo các quy tắc của giao thức cây-chốt. Các giao dịch chỉ đọc đầu tiên có thể chốt bất kỳ hạng mục dữ liệu nào, các giao dịch cập nhật đầu tiên phải chốt gốc.

Chứng tỏ giao thức đảm bảo tính khả tuần tự và không có deadlock

V.7Cho ra ví dụ về các lịch trình có thể dưới giao thức cây nhưng không thể dưới giao thức chốt hai kỳ và ngược lại.

V.8Xét một biến thể của giao thức cây, được gọi là giao thức rừng. CSDL được tổ chức như một rừng. Mỗi giao dịch T phải tuân theo các quy tắc sau:

1. Chốt đầu tiên trong mỗi cây có thể trên bất kỳ hạng mục dữ liệu nào

2. Chốt từ thứ hai trở về sau có thể được yêu cầu chỉ nếu cha của nút được yêu cầu hiện đang bị chốt
3. Các hạng mục dữ liệu có thể được tháo chốt bất kỳ lúc nào
4. Một hạng mục dữ liệu không được chốt lại bởi T sau khi T đã tháo chốt cho nó

Chứng tỏ giao thức rừng không đảm bảo tính khả tuần tự

V.9Khi một giao dịch bị cuộn lại dưới thứ tự tem thời gian, nó được gán một tem thời gian mới. Tại sao không đơn giản để nó giữ lại tem thời gian cũ ?

V.10Trong chốt đa hạt, sự khác nhau giữa chốt tường minh và chốt ẩn là gì ?

V.11phương thức chốt SIX là hữu dụng trong chốt đa hạt. Phương thức exclusive và shared tăng cường không được sử dụng. Tại sao nó lại vô dụng ?

V.12Đối với mỗi một trong các giao thức sau, mô tả sắc thái ứng dụng thực tế gợi ý sử dụng giao thức và sắc thái gợi ý không nên sử dụng giao thức:

1. Chốt hai kỳ
2. Chốt hai kỳ với chốt đa hạt
3. Giao thức cây
4. Thứ tự tem thời gian
5. Hợp lệ
6. Thứ tự tem thời gian đa phiên bản
7. Chốt hai kỳ đa phiên bản

V.13 Chứng tỏ rằng có những lịch trình là có thể dưới giao thức chốt hai kỳ nhưng không thể dưới giao thức tem thời gian và ngược lại.

V.14Với điều kiện nào tránh deadlock ít đắt giá hơn cho phép deadlock rồi phát hiện?

Hệ thống phục hồi
MỤC ĐÍCH Một hệ thống máy tính, cũng giống như các thiết bị cơ - điện khác, luôn có nguy cơ bị hỏng hóc do nhiều nguyên nhân hư đĩa, mất nguồn, lỗi phần mềm v..v... Điều này dẫn đến hậu quả là sự mất thông tin. Vì vậy, hệ quản trị cơ sở dữ liệu phải có các cơ chế đáp ứng lại nguy cơ hệ thống bị hỏng hóc, nhằm đảm bảo tính nguyên tử và tính lâu bền của các giao dịch. Chương này trình bày các nguyên lý của một hệ thống phục hồi nhằm khôi phục CSDL đến một trạng thái nhất quán trước khi xảy ra sự cố. YÊU CẦU Hiểu rõ các sự cố có thể xảy ra trong đời sống của một cơ sở dữ liệu, các nguyên nhân của sự không nhất quán dữ liệu. Hiểu các kỹ thuật phục hồi, các ưu nhược điểm của mỗi kỹ thuật.

PHÂN LỚP HỎNG HÓC:

Có nhiều kiểu hỏng hóc có thể xảy đến với hệ thống, mỗi một trong chúng cần được ứng xử một cách riêng biệt. Trong chương này ta chỉ xét các kiểu hỏng hóc sau:

- Hỏng hóc trong giao dịch: Có hai loại lỗi làm cho giao dịch bị hỏng hóc:1. Lỗi luận lý: Giao dịch không thể tiếp tục thực hiện bình thường được nữa do một số điều kiện bên trong không được thoả. ví dụ như: dữ liệu đầu vào không đúng, không tìm thấy dữ liệu, trào dữ liệu hoặc do việc sử dụng tài nguyên vượt hạn định.2. Lỗi hệ thống: Hệ thống rơi vào trạng thái không mong muốn ví dụ như trạng thái deadlock.

- Hệ thống bị hư hỏng: Có một phần cứng sai chức năng hoặc có một sai sót trong phần mềm cơ sở dữ liệu hay hệ điều hành.

- Đĩa bị hư hỏng: Một khối đĩa bị mất nội dung.

Để hệ thống có thể đề ra được chiến lược phục hồi lỗi phù hợp, trước tiên cần phải xác định các loại hỏng hóc trên các thiết bị lưu trữ dữ liệu. Sau đó, cần xác định những hỏng hóc này ảnh hưởng như thế nào đến nội dung cơ sở dữ liệu. Nhiệm vụ quan trọng sau cùng là đề ra các giải

pháp nhằm đảm bảo tính nhất quán của cơ sở dữ liệu và tính nguyên tử của giao dịch mỗi khi hỏng hóc đã phát sinh. Các giải pháp này thường được gọi là các giải thuật phục hồi (recovery algorithms).

Các giải thuật phục hồi gồm có hai phần:

1. Các hành động được thực hiện trong suốt quá trình hoạt động bình thường của giao dịch nhằm đảm bảo có đầy đủ thông tin cho việc phục hồi sau này.
2. Các hành động được thực hiện sau khi lỗi phát sinh. Nhằm khôi phục nội dung của cơ sở dữ liệu trở về một trạng thái trước đó, và trạng thái này thoã mãn được các yêu cầu về tính nhất quán của cơ sở dữ liệu, tính bền và tính nguyên tử của giao dịch .

CẤU TRÚC LƯU TRỮ:

Như đã xét trong chương II, các hạng mục dữ liệu khác nhau của cơ sở dữ liệu có thể được lưu trên nhiều phương tiện lưu trữ khác nhau. Để nắm được cách thức đảm bảo tính nguyên tử và tính lâu bền của một giao dịch, cần phải có cái nhìn sâu hơn về các loại thiết bị lưu trữ dữ liệu và cách thức truy xuất chúng.

CÁC LOẠI LƯU TRỮ:

- Lưu trữ không ổn định (volatile storage): Thông tin lưu trong thiết bị lưu trữ không ổn định sẽ bị mất khi hệ thống bị hỏng hóc. Ví dụ của thiết bị lưu trữ không ổn định là: bộ nhớ chính, bộ nhớ cache. Sự truy cập đến các thiết bị lưu trữ không ổn định là cực nhanh. Lý do: một là: do tính chất của bộ nhớ cho phép như vậy; hai là: có thể truy xuất trực tiếp các hạng mục dữ liệu chứa trong nó.

- Lưu trữ ổn định (nonvolatile storage): Thông tin lưu trữ trong thiết bị lưu trữ ổn định thường không bị mất khi hệ thống bị sự cố. Tuy nhiên, nguy cơ bản thân thiết bị lưu trữ ổn định bị hỏng vẫn có thể

xảy ra. Ví dụ của thiết bị lưu trữ ổn định là: đĩa từ và băng từ. Trong hầu hết các hệ cơ sở dữ liệu, thiết bị lưu trữ ổn định thường được dùng là đĩa từ. Các loại thiết bị lưu trữ ổn định khác được dùng để lưu trữ phòng hờ (back up) dữ liệu.

- Lưu trữ bền (stable storage): Theo lý thuyết thì thông tin chứa trong thiết bị lưu trữ bền không bao giờ bị mất khi hệ thống bị hư hỏng. Tuy nhiên, trong thực tế, ta khó lòng tạo ra được một thiết bị đạt được tính chất lý tưởng như vậy. Chỉ có giải pháp tăng cường độ bền mà thôi.

THỰC THI LƯU TRỮ BỀN:

Tiêu chí để thực hiện việc lưu trữ bền là nhân bản thông tin cần thiết trong một vài phương tiện lưu trữ ổn địng khác nhau với các phương thức hỏng hóc độc lập và cập nhật các phiên bản thông tin này một cách có tổ chức, sao cho dù có lỗi xuất hiện trong quá trình chuyển dữ liệu, thông tin vẫn không bị hư hại.

- Các hệ thống RAID đảm bảo rằng việc hỏng hóc của một đĩa không gây sự mất dữ liệu. Dạng thức đơn giản và nhanh nhất của RAID là dùng đĩa gương (mirrored disk). Các dạng thức khác giúp tiết kiệm chi phí, nhưng cái giá phải trả là thời gian đọc ghi chậm hơn.

- Tuy nhiên các hệ thống RAID vẫn không đảm bảo được tính an toàn dữ liệu khi gặp phải tai họa như: cháy nổ, lụt lội. Người ta đề nghị một hệ thống lưu trữ mới an toàn hơn hoạt động theo nguyên tắc sau: Sao lưu dữ liệu sang một vài vị trí địa lý khác nhau thông qua mạng máy tính.

Sau đây là cách thức đảm bảo thông tin lưu trữ không bị lỗi trong quá trình đọc ghi dữ liệu:

Việc chuyển một khối dữ liệu giữa bộ nhớ và đĩa có thể dẫn đến kết quả:

- Thành công hoàn toàn: Thông tin được chuyển đến đích an toàn.
- Bị lỗi một phần: Có lỗi xuất hiện trong quá trình chuyển dữ liệu và khối đích chứa thông tin không đúng.
- Bị lỗi hoàn toàn: Lỗi xuất hiện ngay ở giai đoạn đầu của quá trình truyền dữ liệu. Khối đích giữ nguyên như ban đầu.

Nếu có lỗi xuất hiện trong quá trình truyền dữ liệu, hệ thống phải phát hiện được và thực thi thủ tục phục hồi lỗi. Để làm được như vậy, hệ thống phải duy trì hai khối dữ liệu vật lý cho mỗ khối dữ liệu luận lý. (Trong tình huống dùng hệ thống đĩa gương thì hai khối vật lý này ở cùng một địa điểm, trong tình huống dùng hệ thống sao lưu từ xa, hai khối này ở hai địa điểm khác nhau).

Một thao tác ghi dữ liệu được thực thi như sau:

1. Viết thông tin lên khối vật lý thứ nhất.
2. Khi hành động ghi thứ nhất thành công, tiếp tục ghi phần thông tin trên lên khối vật lý thứ hai.
3. Thao tác ghi được coi là thành công khi thao tác ghi thứ hai thành công.

Trong quá trình phục hồi, từng cặp khối vật lý được kiểm tra:

1. Nếu nội dung của cả hai như nhau và không có lỗi có thể phát hiện, khi đó không cần làm gì thêm.
2. Nếu một trong hai khối có lỗi phát hiện được, khi đó thay thế khối bị lỗi bởi nội dung của khối còn lại.
3. Nếu cả hai khối không có lỗi phát hiện được, nhưng nội dung của chúng khác nhau, thay thế khối thứ nhất bởi khối thứ hai.

Yêu cầu phải so sánh từng cặp khối vật lý một đòi hỏi phải mất nhiều thời gian. Người ta có thể cải thiện tình huống này bằng cách lưu vết những thao tác viết khối trong tiến trình thực thi. Khi phục hồi, chỉ những khối nào thao tác ghi ở trong tiến trình thực thi mới cần được đem so

sánh. Giao thức để viết ra một khối đến một site xa tương tự như viết khối trong hệ thống đĩa gương..

TRUY CẬP DỮ LIỆU

Như đã xét trong chương II, hệ cơ sở dữ liệu nằm thường trực trên các thiết bị lưu trữ ổn định (thường là đĩa từ) và thường được phân thành các đơn vị lưu trữ kích thước cố định được gọi là khối (blocks). Khối là đơn vị truyền nhận dữ liệu từ/ra đĩa. Một khối có thể chứa vài hạng mục dữ liệu. Ta giả thiết không có hạng mục dữ liệu nào trải ra trên nhiều hơn một khối.

Các giao dịch nhập (input) thông tin từ đĩa vào bộ nhớ chính và xuất (output) thông tin theo chiều ngược lại. Các thao tác nhập/xuất này được thực hiện theo đơn vị khối. Khối nằm trên đĩa được gọi là khối vật lý (physical block), khối được trữ tạm trong bộ nhớ chính được gọi là khối đệm (buffer block). Vùng bộ nhớ tạm chứa các khối dữ liệu được gọi là vùng đệm đĩa (disk buffer).

Việc di chuyển khối giữa đĩa và bộ nhớ được thực hiện thông qua hai thao tác:

1. Input(B) chuyển khối vật lý B vào bộ nhớ chính.
2. Output(B) chuyển khối đệm B ra đĩa và thay thế cho khối vật lý tương ứng ở đó.

Hình dưới đây sẽ mô phỏng cho hai thao tác này

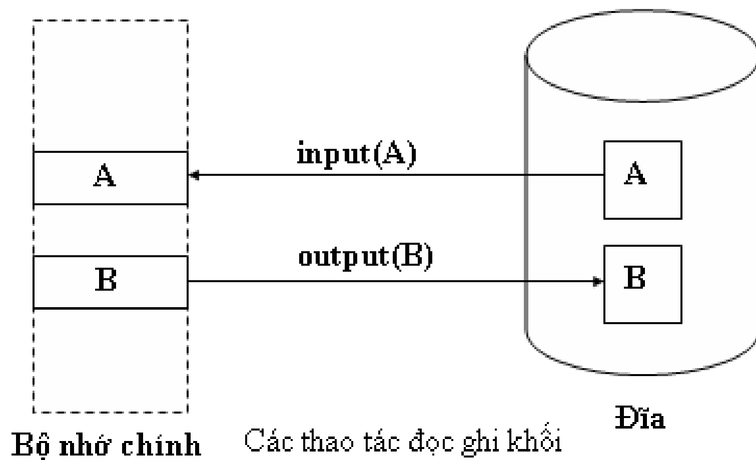


figure VI-

Mỗi giao dịch Ti có một vùng làm việc riêng ở đó các bản sao cùa tất cả các hạng mục dữ liệu được truy xuất và cập nhật được lưu giữ. Vùng làm việc này được tạo ra khi giao dịch khởi động. Nó bị xoá đi khi giao dịch bàn giao (commit) hoặc huỷ bỏ (abort). Mỗi hạng mục dữ liệu x được trữ trong vùng làm việc của giao dịch Ti sẽ được ký hiệu là xi. Giao dịch Ti trao đổi với hệ cơ sở dữ liệu bằng cách chuyển dữ liệu đến/ra vùng làm việc của nó sang vùng đệm của hệ thống.

Hai thao tác dùng để chuyển dữ liệu:

1. read(X) gán giá trị của hạng mục dữ liệu X cho biến cục bộ xi. Thao tác này được thực hiện như sau:

- Nếu khối BX chứa X không có trong bộ nhớ chính thì thực hiện thao tác input(BX).
- Gán cho xi giá trị của X trong khối đệm.

write(X) gán giá trị của biến cục bộ xi cho hạng mục dữ liệu X trong khối đệm. Thao tác này được thực hiện như sau:

- Nếu khối BX chứa X không có trong bộ nhớ thì thực hiện thao tác input(BX).
- Gán giá trị của xi cho X trong vùng đệm BX.

Chú ý rằng cả hai thao tác đều có thể đòi hỏi chuyển một khối từ đĩa vào bộ nhớ chính nhưng không yêu cầu chuyển một khối từ bộ nhớ chính ra đĩa.

Đôi khi một khối đệm bị ghi bắt buộc ra đĩa do bộ quản lý vùng đệm cần không gian bộ nhớ cho các mục đích khác hoặc do hệ cơ sở dữ liệu muốn phản ánh những thay đổi trong khối dữ liệu B trên đĩa. Khi hệ cơ sở dữ liệu thực hiện thao tác Output(B) ta nói nó đã xuất bắt buộc khối đệm B ra đĩa.

Khi một giao dịch cần truy xuất hạng mục dữ liệu X lần đầu, nó phải thực hiện Read(X). Khi đó tất cả các cập nhật đối với X được thực hiện trên xi. Sau khi giao dịch truy xuất X lần cuối, nó thực hiện Write(X) để ghi lại sự thay đổi của X trong CSDL.

Không nhất thiết phải thực hiện thao tác Output(BX) ngay sau khi thao tác write(X) hoàn thành. Lý do là: khối đệm BX có thể còn chứa các hạng mục dữ liệu khác đang được truy xuất. Nếu hệ thống bị hư hỏng ngay sau khi thao tác write(X) hoàn thành, nhưng trước khi thực hiện thao tác Output(BX), giá trị mới của X sẽ không bao giờ được ghi ra đĩa, do đó, nó bị mất!

PHỤC HỒI VÀ TÍNH NGUYÊN TỬ:

Trở lại với ví dụ đơn giản về hệ thống ngân hàng: Giao dịch Ti thực hiện việc chuyển $50 từ tài khoản A sang tài khoản B. Giả sử giá trị ban đầu của các tài khoản A và B là $1000 và $2000. Giả sử hệ thống bị hư hỏng trong khi Ti đang thực thi: sau khi thao tác output(BA) được thực hiện và trước khi thực hiện thao tác output(BB) (BA và BB là hai khối đệm chứa hai hạng mục A và B). Người ta có thể thực hiện một trong hai giải pháp phục hồi sau:

1. Thực hiện lại Ti. Thủ tục này sẽ dẫn đến kết quả: giá trị của A là $900 thay vì phải là $950. Do đó, hệ thống ở trong trạng thái không nhất quán.

2. Không thực hiện lại Ti . Kết quả: giá trị của A và B tương ứng sẽ là $950 và $2000. Hệ thống cũng trong trạng thái không nhất quán.

Vấn đề phát sinh ở chỗ: Ti thực hiện nhiều thao tác sửa đổi nội dung cơ sở dữ liệu, do đó cần nhiều thao tác xuất dữ liệu ra đĩa, nhưng lỗi phát sinh không cho phép tất cả các thao tác xuất dữ liệu hoàn thành.

Giải pháp nhằm đạt được tính nguyên tử là: trước khi thực hiện các thao tác sửa đổi cơ sở dữ liệu, cần ghi ra các thiết bị lưu trữ bền những thông tin mô tả các sửa đổi này. Cụ thể của giải pháp trên sẽ được trình bày trong các phần V.4, V.5 và V.6.

PHỤC HỒI DỰA TRÊN SỔ GHI LỘ TRÌNH (Log-based recovery)

Một cấu trúc thường được dùng để ghi lại những thay đổi trên cơ sở dữ liệu là sổ ghi lộ trình (log). Log là một dãy các mẩu tin lộ trình (log records). Một thao tác cập nhật trên cơ sở dữ liệu sẽ được ghi nhận bằng một log record. Một log record kiểu mẫu chứa các trường sau:

- Định danh giao dịch (transaction identifier): định danh duy nhất của giao dịch thực hiện hoạt động write
- Định danh hạng mục dữ liệu (Data-item identifier): định danh duy nhất của hạng mục dữ liệu được viết (thường là vị trí của hạng mục dữ liệu trên đĩa)
- Giá trị cũ (Old value): giá trị của hạng mục dữ liệu trước khi viết
- Giá trị mới (New value): giá trị hạng mục dữ liệu sẽ có sau khi viết

Có một vài log record đặc biệt mang các ý nghĩa riêng. Bảng sau đây chỉ ra một số loại log record và ý nghĩa của chúng:

LOẠI LOG RECORD	Ý NGHĨA
< Ti start >	Giao dịch Ti đã khởi động.
< Ti, Xj, V1, V2 >	Giao dịch Ti đã thực hiện thao tác ghi trên hạng mục dữ liệu Xj, Xj có giá trị V1 trước khi ghi và nhận giá trị V2 sau khi ghi.
< Ti commit >	Giao dịch Ti đã bàn giao.
< Ti abort >	Giao dịch Ti đã huỷ bỏ.

Mỗi khi một giao dịch thực hiện một thao tác ghi, trước tiên phải tạo ra một log record cho thao tác ghi đó (trong log file), trước khi giao dịch thay đổi cơ sở dữ liệu. Như vậy, hệ thống có cơ sở để huỷ bỏ (undo) một thay đổi đã được làm trên cơ sở dữ liệu bằng cách sử dụng trường Old-value trong log record.

log phải được lưu trong những thiết bị lưu trữ bền. Mỗi một log record mới ngầm định sẽ được thêm vào cuối tập tin log.

CẬP NHẬT TRÌ HOÃN CƠ SỞ DỮ LIỆU (Deferred Database Modification):

Kỹ thuật cập nhật trì hoãn đảm bảo tính nguyên tử của giao dịch bằng cách ghi lại tất cả những sửa đổi cơ sở dữ liệu vào sổ ghi lộ trình (log), nhưng trì hoãn sự thực hiện tất cả các thao tác viết dữ liệu ra đĩa của giao dịch cho đến khi giao dịch bàn giao một phần (partially commits). Nhắc lại rằng: một giao dịch được gọi là bàn giao một phần khi hành động cuối cùng của nó được thực hiện xong. Kỹ thuật cập nhật trì hoãn

được trình bày trong phần này giả thiết rằng các giao dịch được thực hiện một cách tuần tự.

Khi giao dịch bàn giao một phần, thông tin trên log kết hợp với giao dịch được sử dụng trong việc viết trì hoãn. Nếu hệ thống có sự cố trước khi giao dịch hoàn thành việc thực hiện của nó hoặc giao dịch bị bỏ dở khi đó thông tin trên log bị bỏ lơ.

Sự thực thi của một giao dịch được tiến triển như sau:

- Trước khi giao dịch Ti bắt đầu thực hiện, một mẫu tin < Ti start > được ghi ra sổ lộ trình.
- Trước khi Ti thực hiện thao tác write(X), một mẫu tin < Ti, X, V2 > được ghi ra sổ lộ trình.
- Cuối cùng, khi giao dịch Ti bàn giao một phần, mẫu tin < Ti commit > được ghi ra sổ lộ trình.

Khi giao dịch bàn giao một phần, các mẫu tin trong sổ lộ trình kết hợp với giao dịch sẽ được sử dụng để thực hiện việc ghi trì hoãn các hạng mục dữ liệu ra đĩa. Nhà thiết kế phải đảm bảo rằng, trước khi hoạt động ghi hạng mục dữ liệu diễn ra, các mẫu tin log đã được ghi thành công ra các thiết bị lưu trữ bền. Ngoài ra cũng cần để ý: kỹ thuật cập nhật trì hoãn chỉ cần ghi lại giá trị mới của hạng mục dữ liệu (V2) mà thôi.

Để minh hoạ, ta sử dụng ví dụ hệ thống ngân hàng đơn giản. Gọi T0 là giao dịch có nhiệm vụ chuyển $50 từ tài khoản A sang tài khoản B, T1 là giao dịch có nhiệm vụ rút $100 từ tài khoản C. Giả sử giá trị ban đầu của các tài khoản A, B, C là $1000, $2000 và $700. Hành động của T0 và T1 được mô tả như sau:

T0	T1
read(A)A:=A-50write(A)read(B)B:=B+50write(B)	Read(C)C:=C-100write(C)

Giả thiết các giao dịch được thực hiện tuần tự: T0 rồi tới T1. Một phần của sổ lộ trình ghi lại những thông tin liên quan đến hoạt động của hai giao dịch trên được cho trong bảng dưới đây:

<T0 start><T0 ,A, 950><T0 ,B, 2050><T0 commit><T1 start><T1 ,C, 600><T1 commit>

figure VI-

Sau khi có sự cố xảy ra, hệ thống phục hồi sẽ tham khảo sổ lộ trình để chọn ra những giao dịch nào cần được làm lại (redo). Giao dịch Ti cần được làm lại khi và chỉ khi sổ nhật ký có chứa cả hai mẫu tin <Ti start> và <Ti commit>.

Thủ tục làm lại giao dịch Ti như sau:

- redo(Ti) đặt giá trị mới cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch Ti. Các giá trị mới sẽ được tìm thấy trong sổ lộ trình (log).

Hoạt động redo phải đồng hiệu lực (idempotent) có nghĩa là việc thực hiện nó nhiều lần tương đương với việc thực hiện nó một lần.

Trở lại ví dụ vừa nêu, ta có bảng mô tả trạng thái của sổ ghi lộ trình và cơ sở dữ liệu như sau:

LOG	CƠ SỞ DỮ LIỆU

<T0 start><T0 ,A, 950>< T0 ,B, 2050><T0 commit><T1 start><T1 ,C, 600><T1 commit>	A=950B=2050C=600

figure VI-

Sau đây là một số tình huống mô phỏng:

1. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động write(B) của giao dịch T0 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống khởi động trở lại, sẽ không có hành động "thực hiện lại giao dịch" nào cần phải làm, do không có mẫu tin ghi commit nào xuất hiện trong sổ lộ trình. Nghĩa là giá trị của A,B và C vẫn giữ nguyên là $1000, $2000 và $700.
2. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động write(C) của giao dịch T1 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống hoạt động trở lại, thủ tục redo(T0) sẽ được thực hiện do có sự xuất hiện của mẫu tin <T0 commit> trong sổ lộ trình. Sau khi thủ tục này được thực thi, giá trị của A và B sẽ là $950 và $2050.

CẬP NHẬT TỨC THỜI CƠ SỞ DỮ LIỆU (Immediate Database Modification):

Kỹ thuật cập nhật tức thời cho phép các thao tác sửa đổi cơ sở dữ liệu có quyền xuất dữ liệu tức thời ra đĩa trong khi giao dịch vẫn còn ở trong trạng thái hoạt động (active state). Hành động thay đổi nội dung dữ liệu tức thời của các giao dịch đang hoạt động được gọi là "những thay đổi chưa được bàn giao" (uncommitted modifications).

Sự thực thi của một giao dịch được tiến hành như sau:

- Trước khi giao dịch Ti bắt đầu sự thực hiện, một mẫu tin < Ti start > được ghi ra sổ lộ trình.

- Trước khi Ti thực hiện thao tác write(X), một mẫu tin < Ti, X, V1, V2 > được ghi ra sổ lộ trình.
- Cuối cùng, khi giao dịch Ti bàn giao một phần, mẫu tin < Ti commit > được ghi ra sổ lộ trình.

Cần phải đảm bảo rằng, trước khi hoạt động ghi hạng mục dữ liệu diễn ra, các mẫu tin log đã được ghi thành công ra các thiết bị lưu trữ bền. Ngoài ra, cũng cần chú ý là mẫu tin log cho hành động write(X) của giao dịch Ti, tức là mẫu tin <Ti, X, V1, V2> có chứa cả hai giá trị mới (V2) và cũ (V1) của hạng mục dữ liệu X.

Trở lại với ví dụ trong phần V.4.1, ta có một phần của sổ lộ trình liên quan đến các hoạt động của T0 và T1 như sau:

<T0 start><T0 , A, 1000, 950><T0 , B, 2000, 2050><T0 commit><T1 start><T1 , C, 700, 600><T1 commit>

figure VI-

Bảng mô tả trạng thái của sổ ghi lộ trình và cơ sở dữ liệu như sau:

LOG	CƠ SỞ DỮ LIỆU
<T0 start><T0 , A, 1000, 950>< T0 , B, 2000, 2050><T0 commit><T1 start><T1 , C, 700, 600><T1 commit>	A=950B=2050 C=600

figure VI-

Kỹ thuật cập nhật tức thời sử dụng hai thủ tục khôi phục sau lỗi:

- undo(Ti) đặt lại giá trị cũ cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch Ti. Các giá trị cũ sẽ được tìm thấy trong sổ lộ trình (log).
- redo(Ti) đặt giá trị mới cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch Ti. Các giá trị mới sẽ được tìm thấy trong sổ lộ trình (log).

Sau khi lỗi xuất hiện, hệ thống phục hồi tham khảo sổ ghi để quyết định những giao dịch nào cần được làm lại (redo) và những giao dịch nào cần được huỷ bỏ (undo).

- Giao dịch Ti cần được huỷ bỏ khi sổ ghi chứa mẫu tin <Ti start> nhưng không có mẫu tin <Ti commit>.
- Giao dịch Ti cần được làm lại khi sổ ghi có chứa cả mẫu tin <Ti start> lẫn mẫu tin <Ti commit>.

Sau đây là một số tình huống mô phỏng:

1. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động write(B) của giao dịch T0 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống khởi động trở lại, nó sẽ tìm thấy mẫu tin <T0 start> trong sổ ghi, nhưng không có mẫu tin <T0 commit> tương ứng. Do đó giao dịch T0 cần phải được huỷ bỏ. Nghĩa là thủ tục undo(T0) sẽ được gọi và giá trị của A,B và C vẫn giữ nguyên là $1000, $2000 và $700.
2. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động write(C) của giao dịch T1 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống hoạt động trở lại, hai thủ tục redo(T0) và undo(T1) sẽ được thực hiện. Do có sự xuất hiện của các mẫu tin <T0 start>, <T0 commit>, <T1 start> trong sổ lộ trình. Sau khi hai thủ tục này được thực thi, giá trị của A, B, C sẽ là $950,ì $2050 và $700.

ĐIỂM KIỂM SOÁT (Checkpoint):

Khi lỗi hệ thống xuất hiện, hệ thống phục hồi phải tham khảo sổ ghi lộ trình để quyết định những giao dịch nào cần được làm lại và những giao dịch nào cần được huỷ bỏ. Theo nguyên lý thì cần phải tìm kiếm toàn bộ nội dung của sổ ghi để có được quyết định trên.

Hướng tiếp cận trên sẽ gặp phải hai khó khăn lớn:

1. Quá trình tìm kiếm mất nhiều thời gian.
2. Theo các giải thuật vừa nêu, hầu hết các giao dịch cần được làm lại đã ghi những dữ liệu được cập nhật ra cơ sở dữ liệu rồi. Việc làm lại chúng tuy không có hại gì, nhưng lại làm cho tiến trình khôi phục trở nên lâu hơn.

Công cụ “điểm kiểm soát” (checkpoint) được sử dụng để cải thiện hiệu năng của quá trình khôi phục. Trong quá trình hoạt động của mình, hệ thống sẽ duy trì một sổ ghi lộ trình bằng cách sử dụng một trong hai kỹ thuật được giới thiệu trong phần V.4.1 và V.4.2. Ngoài ra, hệ thống còn phải thực hiện một cách chu kỳ các hành động đặt điểm kiểm soát. Hành động này đòi hỏi một dãy các thao tác sau:

1. Xuất ra lưu trữ bền tất cả các mẫu tin ghi nhận lộ trình (log record) đang nằm trong bộ nhớ chính.
2. Xuất ra đĩa tất cả những khối đệm đã được cập nhật.
3. Xuất ra thiết bị lưu trữ bền một log-record <checkpoint>

Các giao dịch sẽ không được phép thực hiện bất kỳ thao tác cập nhật dữ liệu nào (ví dụ như ghi các khối đệm, ghi các mẫu tin log) khi hành động đặt điểm kiểm soát đang được thực hiện.

Sự hiện diện của điểm kiểm soát trong sổ ghi cho phép hệ thống tổ chức quá trình phục hồi tốt hơn. Xét một giao dịch Ti đã bàn giao (commit) trước một điểm kiểm soát. Ta có mẫu tin <Ti commit> xuất hiện trước mẫu tin <checkpoint>. Có nghĩa là tất cả các thay đổi mà Ti đã làm đối với cơ sở dữ liệu phải được thực hiện trước khi người ta đặt điểm kiểm soát trên. Vì vậy, trong giai đoạn phục hồi sau lỗi, người ta không cần phải làm lại (redo) giao dịch Ti.

Dựa trên điểm cải tiến này, ta cải tiến lại các kỹ thuật đã được trình bày trong phần V.4.1 và V.4.2 như sau:

1. Sau khi lỗi hệ thống xuất hiện, hệ thống phục hồi sẽ kiểm tra lại sổ lộ trình (log) để tìm ra giao dịch Ti thoả điều kiện: đó là giao dịch gần đây nhất được khởi động trước điểm kiểm soát gần đây nhất. Qui trình tìm Ti như sau: dò ngược trong sổ ghi lộ trình cho đến khi tìm thấy mẫu tin <checkpoint> đầu tiên. Từ điểm kiểm soát này, lại tiếp tục dò ngược trong sổ ghi cho đến khi tìm thấy mẫu tin <Ti start> đầu tiên. Mẫu tin này chỉ ra giao dịch Ti .
2. Khi đã xác định được giao dịch Ti rồi, các thủ tục undo và redo chỉ được áp dụng cho giao dịch Ti và các giao dịch diễn ra sau Ti. Chúng ta ký hiệu tập những giao dịch vừa nói là T.
3. Với kỹ thuật "Cập nhật tức thời cơ sở dữ liệu", tiến trình phục hồi như sau:

- Với mọi giao dịch Tk T mà không có mẫu tin <Tk commit> trong sổ ghi lộ trình, thực thi undo(Tk).

- Với mọi giao dịch Tk T mà có mẫu tin <Tk commit> trong sổ ghi lộ trình, thực thi redo(Tk).
- Không cần thực thi thao tác undo khi sử dụng kỹ thuật "Cập nhật có trì hoãn cơ sở dữ liệu".

PHÂN TRANG BÓNG (Shadow Paging):

Kỹ thuật "Phân trang bóng" cũng là kỹ thuật cho phép phục hồi sau lỗi, nhưng ý tưởng thực hiện khác với các kỹ thuật dựa trên sổ ghi lộ trình vừa trình bày ở phần trên.

Sau đây là một số khái niệm cần được giải trình:

- Trang (page) là gì? Như đã trình bày ở các phần trước, cơ sở dữ liệu được lưu vào thiết bị lưu trữ không phai thành nhiều khối có kích thước cố định. Người ta gọi những khối này là trang (page).

- Bảng trang và ý nghĩa của nó: Khái niệm trang đã nói được mượn từ lý thuyết về Hệ điều hành. Cách quản lý trang cũng được thừa kế từ đó. Giả sử rằng cơ sở dữ liệu được phân thành n trang và sự phân bố trên đĩa của chúng có thể không theo một thứ tự cụ thể nào cả. Tuy nhiên, phải có cách để tìm ra nhanh và đúng trang thứ i của cơ sở dữ liệu (1   i   n). Người ta dùng bảng trang (được mô phỏng như trong hình 5.2) cho mục đích này. Bảng trang có n đầu vào (entry). Mỗi đầu vào ứng với một trang. Một đầu vào chứa một con trỏ, trỏ đến một trang trên đĩa. Đầu vào đầu tiên chỉ đến trang đầu tiên của cơ sở dữ liệu, đầu vào thứ hai chỉ đến trang thứ hai ...

Ý tưởng then chốt của kỹ thuật “Phân trang bóng” là người ta sẽ duy trì hai bảng trang trong suốt chu kỳ sống của giao dịch, một bảng trang gọi là “bảng trang hiện hành” (current page table), bảng trang còn lại gọi là “bảng trang bóng” (shadow page table). Khi giao dịch khởi động, hai bảng trang này giống nhau. Bảng trang bóng sẽ không thay đổi suốt quá trình hoạt động của giao dịch. Bảng trang hiện hành sẽ bị thay đổi mỗi khi giao dịch thực hiện tác vụ write. Tất cả các tác vụ input và output đều sử dụng bảng trang hiện hành để định vị các trang trong đĩa. Điểm quan trọng khác là nên lưu bảng trang bóng vào thiết bị lưu trữ bền.

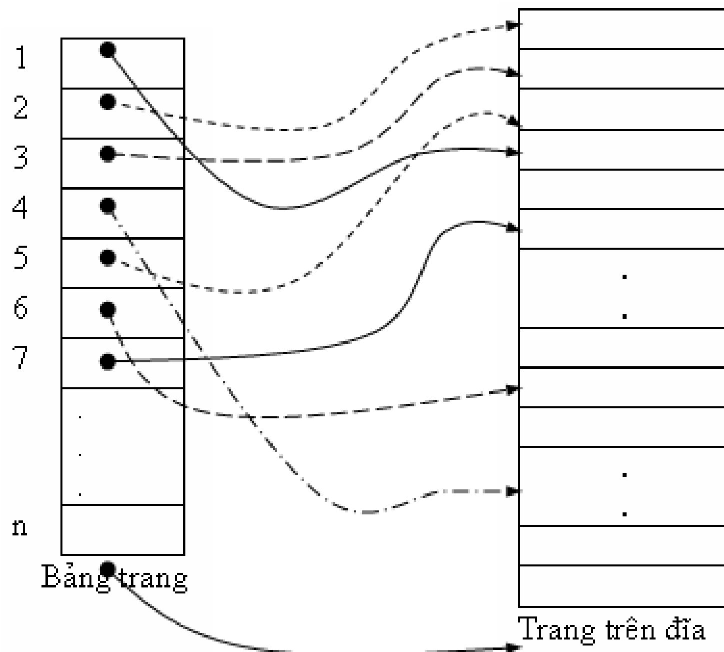


figure VI-

nGiả sử giao dịch thực hiện tác vụ write(X) và hạng mục dữ liệu X được chứa trong trang thứ i. Tác vụ write được thực thi như sau:

1. Nếu trang thứ i chưa có trong bộ nhớ chính, thực hiện input(X).
2. Nếu đây là lệnh ghi được thực hiện lần đầu tiên trên trang thứ i bởi giao dịch, sửa đổi bảng trang hiện hành như sau:

- Tìm một trang chưa được dùng trên đĩa.
- Xoá trang vừa được tìm xong ở bước 2.a khỏi danh sách các khung trang tự do.
- Sửa lại bảng trang hiện hành sao cho đầu vào thứ i trỏ đến trang mới vừa tìm được trong bước 2.a.
- Gán giá trị xi cho X trong trang đệm (buffer page).

Để bàn giao một giao dịch, cần làm các bước sau:

1. Đảm bảo rằng tất cả các trang đệm trong bộ nhớ chính đã được giao dịch sửa đổi phải được xuất ra đĩa.
2. Xuất bảng trang hiện hành ra đĩa. chú ý là không được viết đè lên trang bóng

3. Xuất địa chỉ đĩa của bảng trang hiện hành ra vị trí cố định trong thiết bị lưu trữ bền. Vị trí này chính là nơi chứa địa chỉ của bảng trang bóng. Hành động này sẽ ghi đè lên địa chỉ của bảng trang bóng cũ. Như vậy, bảng trang hiện hành sẽ trở thành bảng trang bóng và giao dịch được bàn giao.

Nếu sự cố xảy ra trước khi hoàn thành bước thứ 3, hệ thống sẽ trở về trạng thái trước khi giao dịch được thực hiện. Nếu sự cố xảy ra sau khi bước thứ 3 hoàn thành, hiệu quả của giao dịch được bảo tồn; không cần thực hiện thao tác redo nào cả. Ví dụ trong hình 5.3 dưới đây mô phỏng lại trạng thái của các bảng trang hiện hành và bảng trang bóng khi giao dịch thực hiện thao tác ghi lên trang thứ tư của cơ sở dữ liệu có 10 trang.

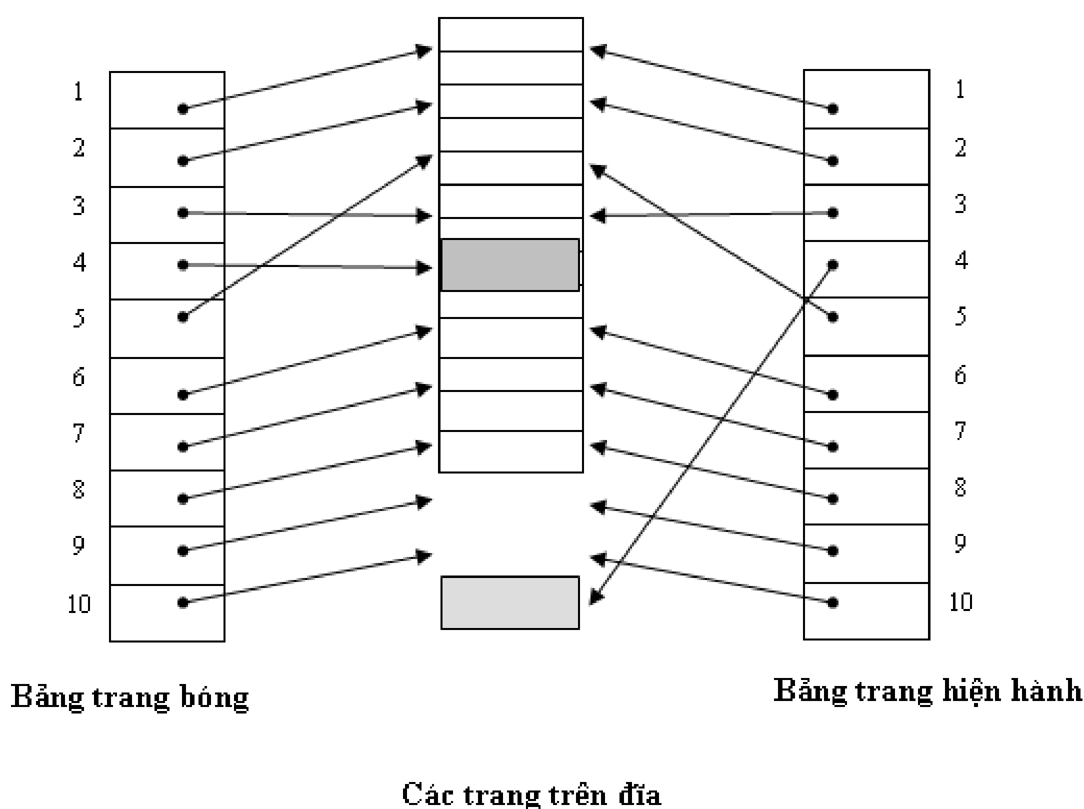


figure VI- Ví dụ về bảng trang bóng và bảng trang hiện hành

Kỹ thuật phân trang bóng có một số điểm lợi hơn so với các kỹ thuật dựa trên sổ ghi:

1. Không mất thời gian ghi ra các log record.
2. Khôi phục sau sự cố nhanh hơn, do không cần các thao tác undo hoặc redo.

Tuy nhiên kỹ thuật phân trang bóng lại có nhiều nhược điểm:

- Tổng phí bàn giao. Xuất nhiều khối ra đĩa: các khối dữ liệu hiện tại, bảng trang hiện hành, địa chỉ của bảng trang hiện hành. Trong kỹ thuật dựa vào sổ ghi, chỉ cần xuất ra các log record, mà thông thường, các log record này vừa đủ chứa trong một khối.

- Sự phân mảnh dữ liệu. Trong chương II có trình bày chiến lược gom cụm vật lý các trang dữ liệu có liên quan với nhau. Sự gom cụm này cho phép việc vận chuyển dữ liệu nhanh hơn. Kỹ thuật phân trang bóng lại đổi vị trí của trang khi trang này bị sửa đổi. Điều này dẫn đến tính gom cụm dữ liệu không còn, hoặc phải dùng các giải pháp gom cụm lại rất mất thời gian.

- Phải thu nhặt rác. Mỗi khi giao dịch bàn giao, các trang chứa giá trị dữ liệu cũ đã bị sửa đổi bởi giao dịch sẽ trở thành không truy xuất được. Vì chúng không thuộc danh sách các trang tự do nhưng cũng không chứa dữ liệu hữu dụng. Ta gọi chúng là “rác”. Cần thiết phải định kỳ tìm kiếm và thêm các trang rác vào trong danh sách các trang tự do. Hành động này được gọi là “thu nhặt rác”.

- Ngoài ra, kỹ thuật phân trang bóng sẽ gặp nhiều khó khăn hơn kỹ thuật dựa vào sổ ghi khi cần được tinh chỉnh để đáp ứng cho yêu cầu phục vụ song song cho nhiều giao dịch. Vì những lý do trên, kỹ thuật phân trang bóng không được sử dụng rộng rãi lắm.

PHỤC HỒI VỚI CÁC GIAO DỊCH CẠNH TRANH

Cho đến bây giờ, ta chỉ xét các kỹ thuật phục hồi áp dụng cho các giao dịch được thực thi tuần tự. Bây giờ chúng ta sẽ tìm cách cải tiến kỹ thuật dựa vào sổ ghi nhằm đáp ứng yêu cầu phục vụ đồng thời cho nhiều giao dịch cạnh tranh. Ý tưởng thực hiện là: Không quan tâm đến số lượng các giao dịch cạnh tranh, hệ thống vẫn sử dụng một vùng đệm đĩa và một sổ ghi lộ trình. Các khối đệm được chia sẻ bởi tất cả các giao dịch. Chúng ta sẽ cho phép việc cập nhật tức thời cơ sở dữ liệu và cho phép một khối đệm có nhiều hạng mục dữ liệu được cập nhật bởi một hoặc nhiều giao dịch.

TRAO ĐỔI VỚI ĐIỀU KHIỂN CẠNH TRANH

Cơ chế phục hồi phụ thuộc rất nhiều vào cơ chế điều khiển cạnh tranh được sử dụng. Để cuộn lại một giao dịch thất bại (failed transaction), người ta phải huỷ bỏ (undo) các cập nhật được thực hiện bởi giao dịch. Giả sử giao dịch T0 phải bị cuộn lại và một hạng mục dữ liệu Q đã bị T0 thay đổi giá trị và cần phải được đặt lại giá trị cũ. Bằng cách sử dụng kỹ thuật dựa vào sổ ghi lộ trình, ta trả lại giá trị cũ cho Q bằng cách sử dụng một log record. Giả thiết lại có giao dịch thứ hai T1 cũng vừa cập nhật Q xong, trước khi T0 bị cuộn lại. Như vậy, sự cập nhật được thực hiện bởi T1 sẽ bị mất đi nếu T0 bị cuộn lại.

Biện pháp khắc phục là: nếu một giao dịch T đã cập nhật một hạng mục dữ liệu Q, thì không một giao dịch nào khác có quyền cập nhật lên hạng mục dữ liệu đó trong khi T chưa bàn giao hoặc chưa bị cuộn lại. Chúng ta có thể đảm bảo yêu cầu trên được thoả bằng cách sử dụng kỹ thuật ”chốt hai kỳ nghiêm ngặt” (strict two-phase locking).

CUỘN LẠI GIAO DỊCH:

Phương pháp để cuộn lại (rollback) một giao dịch Ti, sử dụng sổ ghi, trong môi trường cạnh tranh như sau:

1. Dò ngược sổ ghi lộ trình để tìm ra các log record có dạng <Ti, Xj, V1, V2>.
2. Hạng mục dữ liệu Xj sẽ được trả lại giá trị cũ V1.
3. Việc dò tìm kết thúc khi tìm thấy mẫu tin <Ti start>.

Việc dò ngược sổ ghi lộ [1]eó một ý nghĩa rất quan trọng, do một giao dịch có thể đã cập nhật một hạng mục dữ liệu nhiều hơn một lần. Một ví dụ: Xét một cặp log records như sau:

<Ti, A, 10, 20>

<Ti, A, 20, 30>

Cặp mẫu tin này thể hiện hai hành động cập nhật hạng mục dữ liệu A của giao dịch Ti. Nếu dò ngược sổ ghi lộ trình, A sẽ được trả về giá trị đúng là 10. Ngược lại, A sẽ nhận giá trị sai là 20.

Nếu kỹ thuật strict two-phase locking được sử dụng để điều khiển cạnh tranh, thì việc trả về giá trị cũ cho một hạng mục dữ liệu sẽ không xoá đi những tác động của các giao dịch khác lên hạng mục dữ liệu này.

CÁC ĐIỂM KIỂM SOÁT

Ở phần V.4.3, người ta đã sử dụng điểm kiểm soát (checkpoint) để làm giảm số lượng các log record mà hệ thống phục hồi phải dò tìm trong sổ ghi trong giai đoạn phục hồi sau lỗi. Nhưng, do đã giả thiết là không có cạnh tranh nên giải pháp V.4.3 chỉ xét đến những giao dịch sau trong quá trình khôi phục lỗi:

- Những giao dịch được khởi động sau điểm kiểm soát gần đây nhất.
- Một giao dịch (nếu có) đang trong trạng thái hoạt động (active) tại thời điểm người ta đặt điểm kiểm soát gần đây nhất.

Tình huống càng phức tạp khi các giao dịch được thực thi cạnh tranh. Có nghĩa là tại thời điểm đặt điểm kiểm soát, có thể có nhiều giao dịch đang

ở trong trạng thái hoạt động.

Trong một hệ thống xử lý các giao dịch cạnh tranh, ta yêu cầu rằng: một mẫu tin ghi dấu kiểm soát (checkpoint log record) phải có dạng như sau:

<checkpoint L>

Trong đó L là danh sách các giao dịch đang hoạt động tại thời điểm đặt điểm kiểm soát.

Một lần nữa, ta qui ước rằng: khi hành động đặt điểm kiểm soát đang diễn ra, các giao dịch không được phép thực hiện bất kỳ thao tác cập nhật dữ liệu nào cả trên các khối đệm lẫn trên sổ ghi lộ trình.

Tuy nhiên, qui ước trên lại gây phiền toái, bởi vì các giao dịch phải ngừng hoạt động khi đặt điểm kiểm soát. Một kỹ thuật nâng cao giải quyết điểm phiền toái này là "Điểm kiểm soát mờ" (fuzzy checkpoint).

PHỤC HỒI KHỞI ĐỘNG LẠI (Restart Recovery)

Khi hệ thống phục hồi sau lỗi, nó tạo ra hai danh sách: undo-list bao gồm các giao dịch cần phải huỷ bỏ và redo-list bao gồm danh sách các giao dịch cần được làm lại.

Qui trình tạo lập hai danh sách redo-list, undo-list được thực hiện như sau:

1. Đầu tiên, chúng sẽ rỗng.
2. Dò ngược sổ ghi lộ trình, kiểm tra các mẫu tin cho đến khi tìm được mẫu tin <checkpoint> đầu tiên:

- Với mỗi mẫu tin được tìm thấy theo dạng <Ti commit>, ta thêm Ti vào trong redo-list.
- Với mỗi mẫu tin được tìm thấy theo dạng <Ti start>, nếu Ti không thuộc redo-list thì thêm Ti vào trong undo-list.

- Khi tất cả các log record đã được xem xét, ta kiểm tra danh sách L trong mẫu tin <checkpoint L>. Với mọi giao dịch Ti trong L, nếu Ti không thuộc redo-list thì thêm Ti vào undo-list.

Khi hai danh sách redo-list, undo-list được thiết lập xong, tiến trình phục hồi được tiến hành như sau:

1. Dò ngược lại sổ ghi và thực hiện thủ tục undo đối với mỗi log record thuộc về giao dịch Ti trong undo-list. Các log record của các giao dịch nằm trong danh sách redo-list sẽ bị bỏ qua trong giai đoạn này. Việc dò ngược sẽ ngưng khi mẫu tin <Ti start> được tìm thấy cho mọi giao dịch Ti thuộc danh sách undo-list.
2. Định vị mẫu tin <checkpoint L> gần đây nhất trong log.
3. Dò sổ ghi theo chiều xuôi bắt đầu từ mẫu tin <checkpoint L> gần đây nhất và thực hiện thủ tục redo đối với mỗi log record thuộc về giao dịch Ti nằm trong danh sách redo-list. Trong giai đoạn này, bỏ qua các log record của các giao dịch thuộc danh sách undo-list.

Việc xử lý ngược ở bước 1 là rất quan trọng, nhằm đảm bảo kết quả trả về của cơ sở dữ liệu là đúng.

Sau khi tất cả các giao dịch trong danh sách undo-list bị huỷ bỏ, tất cả các giao dịch trong danh sách redo-list sẽ được làm lại. Sau khi tiến trình phục hồi thành công, xử lý giao dịch được tiếp tục.

Việc thực hiện huỷ bỏ các giao dịch trong undo-list trước khi làm lại các giao dịch trong redo-list có ý nghĩa rất quan trọng. Nếu làm ngược lại, vấn đề sau sẽ phát sinh: Giả sử hạng mục dữ liệu A có giá trị khởi đầu là 10. Giao dịch Ti đổi A thành 20 sau đó Ti bị huỷ bỏ. Sau đó, một giao dịch khác Tj cập nhật A thành 30. Đến đây thì hệ thống bị lỗi ngừng hoạt động. Hiện trạng của sổ ghi tại thời điểm hệ thống bị lỗi như sau:

<Ti, A, 10, 20>

<Tj, A, 10, 30>

<Tj commit>

Nếu thực hiện redo trước, A sẽ được đặt giá trị 30. Sau đó thực hiện undo, A sẽ được đặt giá trị 10, mà giá trị này sai. Giá trị cuối cùng của A phải là 30.

ĐIỂM KIỂM SOÁT MỜ (fuzzy checkpoint):

Kỹ thuật fuzzy checkpoint cho phép các giao dịch được cập nhật dữ liệu trên các khối đệm khi checkpoint-record đã được viết xong nhưng trước thời điểm các khối đệm đã sửa đổi được ghi ra đĩa.

Ý tưởng thực hiện fuzzy checkpoint như sau: Thay vì phải dò ngược sổ ghi để tìm mẫu tin checkpoint, ta sẽ lưu vị trí của mẫu tin checkpoint cuối cùng trong sổ ghi vào một chỗ cố định trong đĩa gọi là last_checkpoint. Tuy nhiên, thông tin này sẽ không được cập nhật khi một mẫu tin checkpoint được ghi ra đĩa. Thay vào đó, trước khi một mẫu tin checkpoint được ghi ra sổ ghi, ta tạo ra một danh sách các khối đệm bị sửa đổi. Thông tin last_checkpoint được cập nhật chỉ sau khi tất cả các khối đệm bị sửa đổi được ghi ra đĩa.

last_checkpoint chỉ được dùng cho mục đích undo.

BÀI TẬP CHƯƠNG VI

1. Trình bày các điểm khác nhau giữa 3 kiểu lưu trữ: lưu trữ không ổn định, lưu trữ ổn định và lưu trữ bền theo tiêu chuẩn đánh giá là chi phí cài đặt.
2. Thực tế, lưu trữ bền không thể thực hiện được. Giải thích tại sao? Hệ cơ sở dữ liệu giải quyết vấn đề này như thế nào?
3. So sánh các kỹ thuật cập nhật tức thời và cập nhật có trì hoãn trong sơ đồ phục hồi dựa vào sổ ghi lộ trình theo các tiêu chuẩn: tính dễ cài đặt và tổng chi phí thực hiện.
4. Giả sử rằng kỹ thuật cập nhật tức thời được sử dụng trong hệ thống. Bằng ví dụ, hãy chỉ ra rằng: tình trạng không nhất quán dữ liệu sẽ xảy ra nếu các log record không được ghi ra thiết bị lưu trữ bền trước khi giao dịch bàn giao (commit).

5. Giải thích mục đích của cơ chế điểm kiểm soát (checkpoint). Hành động đặt điểm kiểm soát nên được thực hiện theo chu kỳ bao lâu là hợp lý?
6. Khi hệ thống phục hồi sau lỗi, nó xây dựng 2 danh sách: undo-list và redo-list. Giải thích tại sao các log record của các giao dịch trong danh sách undo-list phải được xử lý theo thứ tự ngược, trong khi những log record trong danh sách redo-list lại được xử lý theo chiều xuôi?
7. So sánh sơ đồ phục hồi phân trang bóng và sơ đồ phục hồi bằng sử dụng sổ ghi lộ trình theo tiêu chuẩn: tính dễ cài đặt và tổng chi phí thực hiện.
8. Giả sử một cơ sở dữ liệu có 10 khối đĩa liên tiếp (khối 1, 2, 3, ..., 10). Hãy thể hiện trạng thái của buffer và thứ tự vật lý có thể có của các khối sau các thao tác cập nhật sau, giả sử: kỹ thuật phân trang bóng được sử dụng, buffer trong bộ nhớ chỉ đủ chứa 3 khối, chiến lược quản lý buffer là LRU (Least Recently Used)

Đọc khối 3
Đọc khối 7
Đọc khối 5
Đọc khối 3
Đọc khối 1
Sửa đổi khối 1
Đọc khối 10

Sửa khối 5